High Availability iRODS System (HAIRS)

Yutaka Kawai[*] *Adil Hasan*[#]
[*] Computing Research Center, High Energy Accelerator Research Organization (KEK)
[#] School of English, University of Liverpool

Abstract

The integrated Rule Oriented Data Management System (iRODS) is a policy-driven data management system that is starting to be used by projects with large data volume requirements that require a highly available system. In this paper we describe an approach to provide a Highly Availability load-balanced iRODS System (HAIRS). We also describe the advantages and disadvantages of the approach and future work.

Index Keyword Terms— High Availability, UltraMonkey, PgPool, Director, ldirectord, ipvsadm

1. Introduction

The integrated Rule Oriented Data Management System (iRODS) [6] is an open-source, policy-driven distributed data management system developed by the Data Intensive Cyber Environments group that insulates its' users from changes to the physical components of the system. Interaction with data stored in the iRODS system is done using logical file-names and storage names. The iRODS system takes care of the translation from the logical to the physical name. Changes to the physical location of a file only requires the logical-to-physical file mapping to be updated.

Changes to the physical storage resource require an update to the logical-to-physical storage resource mapping and, if required, the implementation of a new iRODS driver that is able to translate iRODS file commands to those used by the physical storage resource. In this way iRODS provides a uniform interface to heterogeneous storage resources. In addition to a virtual file-system iRODS also provides the possibility to impose a series of directives (collective called policies, or rules) on the data stored. In keeping with the iRODS philosophy the rules are defined in a high-level, fully-featured language with each step of the rule implemented as a C-base service (termed a micro-service).The rule is insulated from changes to the underlying micro-services.

An iRODS system consists of one iRODS server that communicates directly with the iRODS Metadata Catalog (iCAT) database and an iRODS server running on each storage resource. All iRODS servers require an iRODS rule engine that executes the triggered rules. An iRODS system can be federated with another iRODS system providing seamless access to data stored in a remote iRODS system. The iRODS is starting to be used by projects with large numbers of users and with large data volume requirements in Japan [10, 11, 14], France [10, 4], the USA [10, 18] and Australia [1]. Such projects operate in an 'always-on' mode and cannot tolerate a failure in accessing the data. Within iRODS a failure of a single storage resource can be mitigated by replicating the data over more than one resource. But, the iCAT and the iCAT-enabled iRODS server remain as a single point of failure. If the iCAT database is down, or if the iCAT enabled server is offline the iRODS system cannot be used.

In Section 2 we describe the approach of database replication to mitigate against iCAT server failure and in Sections 3 we describe the approaches to mitigate against iCAT-enabled server failure. Section 4 describes some of the tests we performed in order to determine the impact of the approach and Section 5 outlines future work.

2. Redundant iCAT

The iRODS Metadata Catalogue (iCAT) contains all the information necessary to manage files stored in iRODS. The iCAT is implemented as a set of tables in a PostgreSQL, ORACLE or MySQL database. Only one iCAT exists per iRODS system and, as such, forms a single point of failure. Implementing database replication techniques can eliminate this critical point.

The Australian Research Collaborative Service has implemented PostgreSQL database replication for the iCAT using PgPool [9]. The procedure essentially requires setting up two iCAT PostgreSQL databases that are replicated via PgPool as shown in figure 1. The iCAT databases are interfaced to two iRODS servers A and B, and clients can connect to either server. Any

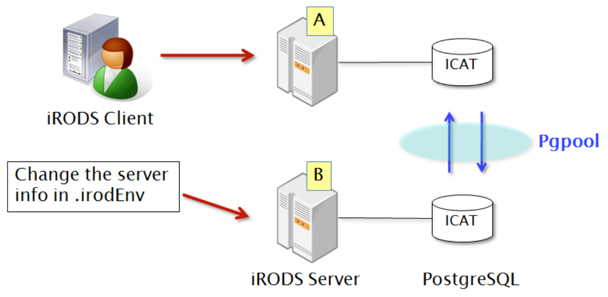

Figure 1. iRODS High Availability using PgPool.

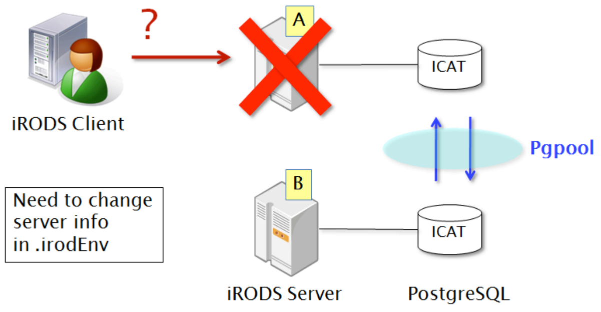

Figure 2. Failure situation: iRODS server A is down.

changes to either iCAT are automatically replicated to the other iCAT.

A similar approach can be used for an iRODS that uses MySQL [16], while Oracle provides its own mechanisms for replicating databases [15].

This approach is extremely useful for creating a fault-tolerant iCAT although it requires the client to actively know which iRODS-enabled ICAT server they are connected to and to alter their configuration if their default server is down (see figure 2). In Section 3 we describe an approach that addresses this problem.

3. Redundant iCAT Enabled iRODS Server

An iRODS consists of only one iRODS server that interfaces to the iCAT. Like the iCAT this server is also a critical component of the iRODS and redundancy of this server would eliminate this single point of failure. By making use of a load-balancer application [12] one can create a redundant pool of servers with a single point of entry for the client application. In this way the client does not need to remember which set of servers belong to the pool and new servers can be added to the pool as required allowing the system to scale with increasing load.

There are a number of load-balancers available that enable a redundant system to be built, these split along hardware or software lines. For example, the CISCO CATALYST 6500 [2] hardware component, can do Layer 4 switching and has load-balancing algorithms. Hardware load-balancers are high-performance, robust and tend to be expensive. Examples of software load-balancers are HAProxy [3] that supports http, ssh etc protocols and Ultra Monkey [17] that provides support for a wide range of protocols. At the time of writing HAProxy does not provide support for simple-TCP based protocols on which the iRODS protocol is based and so Ultra Monkey was used in this study.

The approach used in this paper is to make use of a software load-balancer and adapt it to provide a pool of iCAT enabled iRODS servers that are mapped to a virtual server which the client connects to. This approach ensures that if one server is unavailable the client will be directed to the next available server.

Ultra Monkey is a Linux-based load-balancer that makes use of Linux Virtual Server [13] to provide a fast load-balancer implemented as the Linux Director as shown in figure 3. The Linux Director ideally runs on a separate server and essentially contains a list of real servers which are regularly polled. Clients connect to the director which then forwards requests to the least loaded server. If one of the servers is overloaded or down the client is automatically redirected to another server in the pool. The Linux Director is only used to establish a connection between the client and the least-loaded, working iRODS server. Once the connection has been established iRODS takes over to complete the interaction. This ensures that the extra cost (in time) due to the Linux Director is minimal.

In this way the iRODS system can scale with increasing load as new iRODS servers can be added to the pool as needed without the client needing to update their configuration. The following sections describe the load-balancer setup used in this work.

3.1. Network Configuration

The network configuration of the load-balancer is shown in figure 4 and in tables 1 and 2. The Linux Director is installed on a separate server and behaves as a virtual iRODS server that maps the client request to a real iRODS server (it behaves effectively as a Network Address Translation device). The Linux Director and the iRODS servers need to be in the same domain as the load-balancer cannot span different domains (i.e. the Linux Director cannot load-balance over a pool of servers that are located in different administrative domains).

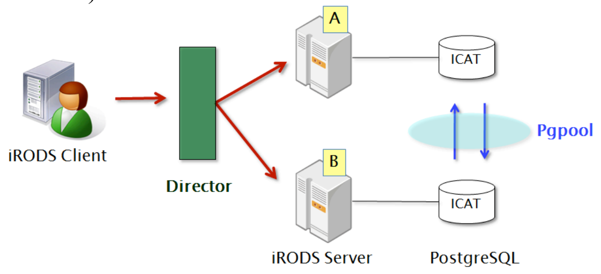

Figure 3. Solution using Director.

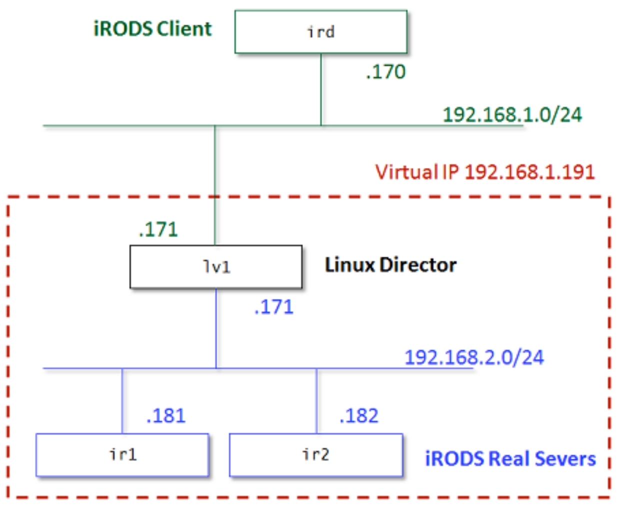

Figure 4. Example: Network Configuration

IP address	Description
192.168.1.171	Linux Director for 192.168.1.0/24 network
192.168.1.191	Virtual Server
192.168.1.170	iRODS Client

Table 1. Network 192.168.1.0/24

IP address	Description
192.168.1.171	Linux Director for 192.168.2.0/24 network
192.168.1.191	iRODS Real Server 1
192.168.1.170	iRODS Real Server 2

Table 2. Network 192.168.2.0/24

3.2. Linux Director Installation

The Linux Director was installed on a CentOS5 Linux server. In addition to the Linux Director server application the following applications need to be installed (more details can be found on the Ultra Monkey web site [17]):

- heartbeat: runs on the Linux Director server and polls the iRODS servers to determine their load.
- heartbeat-ldirectord: interfaces the heartbeat application to the Linux Director to allow clients to be directed to the least loaded server.
- heartbeat-pils: plug-in interface application to interface to the Linux Director.
- heartbeat-stonith: used to remotely power down a node in the pool.
- Ipvsadm: administers IP virtual server services offered by the Linux kernel.
- Libnet: utilities to help with managing network packets.

```
<MsgHeader_PI>
<type>RODS_VERSION</type>
<msgLen>182</msgLen>
<errorLen>0</errorLen>
<bsLen>0</bsLen>
<intInfo>0</intInfo>
</MsgHeader_PI>
<Version_PI>
<status>-4000</status>
<relVersion>rods2.1</relVersion>
<apiVersion>d</apiVersion>
<reconnPort>0</reconnPort>
<reconnAddr></reconnAddr>
<cookie>0</cookie>
</Version_PI>
```

Figure 5. Routine strings iRODS server returns

There are several things to care about when installing the Linux Director for an iRODS system. The Linux Director daemon ldirectord reads its configurations from the configuration file ldirectord.cf which, by default is be installed in /etc/ha.d. The configuration file contains the list of iRODS servers that the Linux Director must map the client to. In order for Ultra Monkey to work with the iRODS protocol the "service" flag in the ldirectord.cf file should be "simpletcp". The iRODS server returns routine messages whenever it receives any message from a client (figure 5). Therefore, the "request" flag in the ldirectord.cf can contain any client request (the iRODS ils client command was used as this interacted with the metadata catalogue and ensured the whole system was functioning). The "receive" flag should be specified as "RODS VERSION" which is a part of the iRODS server response. An example of the ldirectord.cf file is shown in figure 6.

```
checktimeout=10
checkinterval=2
autoreload=yes
logfile="/var/log/ldirectord.log"
logfile="local0"
quiescent=no

virtual=192.168.1.191:1247
        real=192.168.2.181:1247 masq
        real=192.168.2.182:1247 masq
        protocol=tcp
        service=simpletcp
        request="test"
        receive="RODS_VERSION"
        scheduler=lc
        checktype=negotiate
        netmask=255.255.255.255
```

Figure 6. An example of ldirectord.cf.

The ipvsadm (Linux Virtual Server administration) command can have one of ten types of scheduling-method [5]. It is configured by the flag "scheduler", table 3 shows a list of the scheduling methods ldirectord can configure. The ldirectord configuration in the figure 6 specifies "lc" to assign more jobs to real servers with fewer active jobs.

Scheduler Flag	Scheduling Method
rr	Round Robin
wrr	Weighted Round Robin
lc	Least-Connection
wls	Weighted Least-Connection
lblc	Locality-Based Least-Connection
lblcr	Locality-Based Least-Connection with Replication
dh	Destination Hashing
sh	Source Hashing
sed	Shortest Expected Delay
nq	Never Queue

Table 3. ipvsadm scheduling-method Algorithm.

4. Tests

In this section we describe the tests carried out to determine the performance impact of the load-balancer. The first test addresses the impact of the load-balancer on the transfer of large files and the second concerns the overhead the load-balancer places on client interaction with the iRODS server. Both tests made use of the client C-based iRODS utilities ("icommands") that form part of the iRODS suite [8].

4.1. Large File Transfer

The "iput" command is used to store a file into an iRODS system. By default, if the file size is larger than 32 MB, iput performs the transfer in parallel [7]. In this case the data transfer is carried out directly between the physical resource and the client as shown in figure 7:

1. Client issues iput with a large file.
2. Server A finds the physical location to store the file.
3. Server A directs the other iRODS Server C with the physical storage to open parallel I/O ports.
4. File transfer starts between Client and Server C.

Redundancy of iRODS storage servers is provided by replicating data over more than one storage server and so the load-balancer does not need to be configured to provide redundancy for these servers; only for the iRODS iCAT-enabled server. This greatly simplifies the configuration as shown in figure 8 as the ports that the large file transfers occur on do not need to be mapped in the Linux Director configuration.

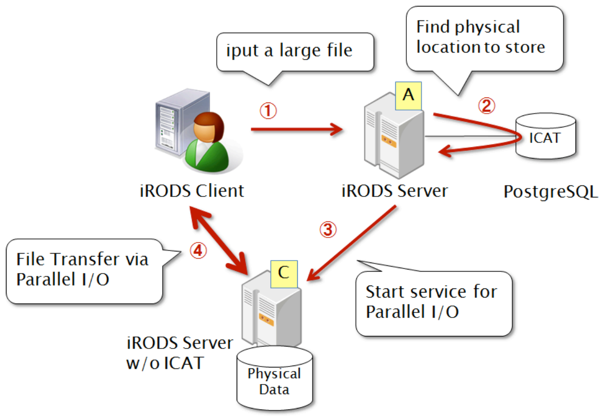

Figure 7. Large file transfer: Normal case.

The configuration is almost exactly as in figure 7 except that the Linux Director forwards the client connection to an iRODS server which then forwards the request to the target storage system. This setup limits the complexity of the configuration of the Linux Director and eliminates the impact of the load-balancer on the transfer of large files. In our tests files of 1GB in size were successfully stored in iRODS with the client.

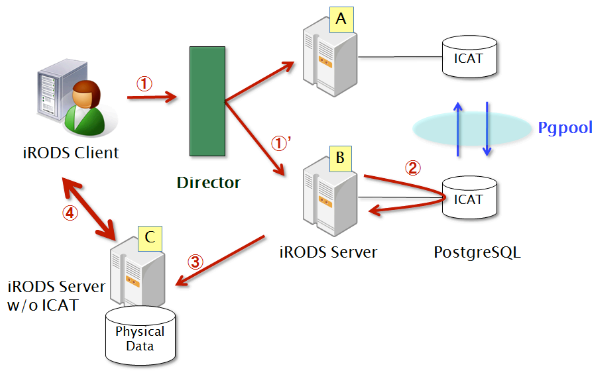

Figure 8. Large file transfer: The case using director.

4.2. Load-balancer Overhead

The iRODS suite contains a package for performing concurrent tests on an iRODS system. This package was used to understand the overhead the load-balancer places on an iRODS system. The concurrent test sequentially executes several icommands, iput (to store data), imeta (to query the metadata catalogue), iget (to retrieve data), and imv (to move data from one iRODS resource to another). The concurrent tests were performed for 1, 10, 50 and 100-1000 clients. The network configuration is the same as the example in the previous sections (figure 4). Physically, all the iRODS servers are Xen virtual machines on the same physical machine and the only iRODS client is on the different physical machine. This can have a non-trivial and noticeable effect on the results of the tests.

Three series of tests were performed to understand the impact of the load-balancer:

case1: Normal case. The iRODS client directly accesses one iRODS server.
case2: Using a director. The iRODS client accesses one iRODS server through the Linux Director.
case3: Load sharing case. The iRODS client accesses two iRODS servers through the Linux Director.

In order to get the average values, the concurrent-test program is executed three times for each test. The figure 9 shows the results of the tests. The case 2 is about 10% slower than the normal case 1 so the impact of the speed performance by using director should be considered. However, while considering optimization of Director implementation, controlling tradeoff between access speed and benefits of high availability becomes practical.

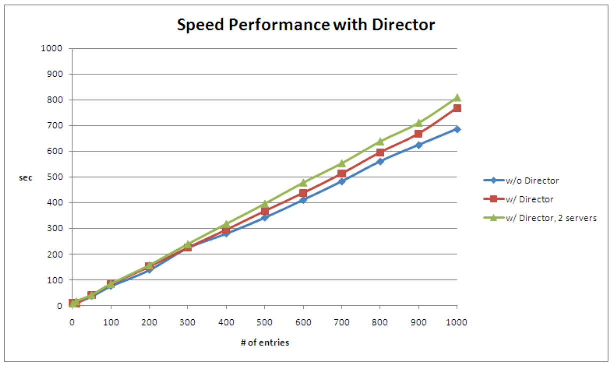

Figure 9. Speed Performance Test Results.

5. Conclusion and Future work

This paper has described how a highly available iRODS system can be implemented with a load-balancer with negligible impact to the client. The impact of the load-balancer on the performance of the iRODS system is minimal and should be considered in the case where a highly available system is needed. Although the approach described was for the Ultra Monkey load-balancer we believe the same approach can be used for any other load-balancer. In addition this approach can also result in a highly scalable iRODS system that can grow with increasing load.

One area that we consider to be limiting is the restriction of the redundant iRODS servers to be within the same domain. A truly high availability system would try to eliminate domain-specific problems by having a pool of servers that span multiple domains. This is an area we are looking at addressing in the future. We are also looking at applying the concept of HAIRS to other catalog services such as the RNS (Resource Namespace Service) application, Gfarm (Grid Data Farm), etc.

6. Acknowledgment

The authors would like to thank to Prof.Takashi Sasaki and Yoshimi Iida. Prof.Sasaki coordinated the study. Ms.Iida gave us valuable support for the iRODS setup in KEK. Adil Hasan is also grateful to the KEK institute for their kind hospitality and to the KEK Short-term Visiting Scientist program financial support during the course of this study.

7. References

[1] Australian Research Collaboration Service. Online. http://projects.arcs.org.au/trac/podd/wiki/iRODS.
[2] CISCO CATALYST 6500 SERIES CONTENT SWITCHING MODULE. Online. http://www.cisco.com/en/US/products/hw/modules/ps2706/products_data_sheet09186a00800887f3.html.
[3] HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer. Online. http://haproxy.1wt.eu/.
[4] IN2P3 – National Institute of Nuclear Physics and Particle Physics. Online. http://cc.in2p3.fr/?lang=en.
[5] ipvsadm(8) - Linux man page, scheduler option. Online. http://linux.die.net/man/8/ipvsadm.
[6] iRODS – the Integrated Rule-Oriented Data System. Online. http://www.irods.org.
[7] iRODS file transfer. Online. https://www.irods.org/index.php/iRods_file_transfer.
[8] iRODS icommands. Online. https://www.irods.org/index.php/icommands.
[9] iRODS Master/Slave Replication with pgpool. Online. https://projects.arcs.org.au/trac/systems/wiki/DataServices/iRODS_Replication_Pgpool.
[10] Projects Using and Developing iRODS. Online. http://www.diceresearch.org/DICE_Site/iRODS_Uses.html.
[11] KEK – High Energy Accelerator Research Organization, KEK. Online. http://www.kek.jp/intra-e/index.html.
[12] Load balancing (computing). http://en.wikipedia.org/wiki/Load_balancing_%28computing%29.
[13] The Linux Virtual Server. Online. http://www.linuxvirtualserver.org/.
[14] Lyon-KEK. Online. https://www.irods.org/index.php/Lyon-KEK.
[15] P. McElroy and M. Pratt. Oracle Database 11g: Oracle Streams Replication. Technical report, Oracle, 2007. http://www.oracle.com/technology/products/dataint/pdf/twp_streams_replication_11gr1.pdf.
[16] MySQL Master Master Replication. Online. http://www.howtoforge.com/mysql_master_master_replication.
[17] UltraMonkey, Load Balancing and High Availability Solution. Online. http://www.ultramonkey.org/.
[18] Information Technology Services for The University of North Carolina at Chapel Hill. Online. http://its.unc.edu/its/index.htm.