Research Data Management at RUG

Centre for Information Technology
The Research Data Management System at the University of Groningen (RUG RDMS): architecture, solution engines and challenges

A. Tsyganov, S. Stoica, M. Babai, V. Soancatl-Aguilar, J. Mc Farland, G. Strikwerda, M. Klein, V. Boxelaar, A. Pothaar, C. Marocico, J. van den Buijs

(rdms-support@rug.nl)
RUG RDMS

Facts and figures

Key figures
- Founded in 1614
- 11 faculties (1 in the Frisian capital of Leeuwarden)
- 6,250 fte staff (including University Medical Center Groningen, UMCG)
- More than 120 nationalities
- 140,000 alumni
- 1,000 million EUR budget

Education (2020)
- 34,000 students
- 8,250 international students (24%)
- > 120 English-taught Master's degree programmes
- > 45 Bachelor's degree programmes, of which >35 taught in English

Research (2020)
- 3,600 fte academic staff (39% international)
- 425 full professors
- 4,350 PhD candidates (51% international)
- 591 PhD theses
- 8,000 research publications (dissertations not included)
- 18 patent applications

Excellent prizes and grants (2015-2020)
- Nobel Prize for Ben Feringa

https://www.rug.nl/about-ug/profile/facts-and-figures/
RUG RDMS: Agenda

- Storage architecture
- Application design
- Web Interface
- iRODS rules
- Custom policies engine
- Metadata templates
- Auditing
- What’s next?
- Q&A
Storage Architecture
RUG RDMS: Storage Architecture

- our own "Google Drive" for Research-data (keep data on-premises)
- object store instead of old-school filesystems
- self service: user can set ACL's on files without helpdesk
- self service: user can update/add metadata to files
- home-data, group-data, project-data
- files/access can be audited (rabbitmq/kibana/ELK-stack)
- files are replicated (2 copies, spread between 3 locations)
RUG RDMS: Storage Architecture

Network: all systems are part of data.rug.nl
- host-based firewalls
- central firewall
- only rug-net/umco network access allowed
- nothing public (yet)
RUG RDMS: Storage Architecture

3 'Big' storage-servers (located in 3 different DC's)

Ubuntu/ZFS: fancy/hip filesystem (data-deduplication, compression, 'self'-healing, software RAID

```
root@gargantua0:~# zpool status | grep sd | wc -l
90

root@gargantua0:~# zpool list
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>ALLOC</th>
<th>FREE</th>
<th>EXPANDSZ</th>
<th>FRAG</th>
<th>CAP</th>
<th>DEDUP</th>
<th>HEALTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool0</td>
<td>616T</td>
<td>97.1T</td>
<td>519T</td>
<td>-</td>
<td>0%</td>
<td>15%</td>
<td>1.00x</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>
RUG RDMS: Storage Architecture

IRODS Zones:

- rugZone: big zone for current projects
- umcgZone: dedicated zone for UMCG (storage on datahandeling)
- sramZone: dedicated (test) zone for SRAM (external users test)
- testZone: dedicated test zone for user-test/workshops
- devZone: dedicated dev zone for developers/playground

More/extra Zones are VM’s, so they can run on the same central Infrastructure.
RDMS backend should survive...

- broken disks (hot-swappable)
- broken volume (files are replicated)
- broken server(s)
- 1 downed data-centre
- small scale network outage
- maintenance...
- resc-vm’s are small (move/migrated if needed)
- scalable workload (load-balancers, multi-site)
- High Performance Computing workloads
RUG RDMS: Storage Architecture

Authentication:
- PAM-based authentication stack
- login:email/university-password
- RUG-LDAP (rug-users)
- SRAM-LDAP (for external users)
RUG RDMS: Storage Architecture

1. **SRAM LDAP@surf**
   - **URL**: https://sbs.sram.surf.nl
   - **Function**: Portal to create/modify SRAM-users

2. **data-manager@rug**
   - **Function**: Manage SRAM users

3. **PAM based Linux LDAP NSLDCD**
   - **Function**: Authentication service

4. **RDMS Icat's**
   - **Function**: Internal database

5. **'normal' user ldap@rug**
   - **Function**: Internal user access

6. **'enriched' (extra attributes added, password field)**
   - **Function**: Enhanced user record

7. **SRAM container@ldap-rug**
   - **Function**: SRAM user container

8. **VPN**
   - **Function**: Secure network connection

9. **RDMS Frontend portal**
   - **Function**: User interface to access RDMS

10. **External user**
    - **Function**: External user access
RUG RDMS: Storage Architecture

Security best-practices:
- no root ssh login on systems
- SSL everything
- only 2 admins (admin-group) can ‘sudo’
- ssh-login only from admin-network (BWP-lan)
- host-based firewall (iptables)
- network based central firewall (Palo Alto)
- stealth (intruder detection/file integrity)
- fail2ban on webdav
- todo/nice-to-have: fail2ban on iRODS
- monthly security-audit on systems by sec team
- external company pentesting (yearly)
Application design
RUG RDMS: Application design

- Modularity
  - Different applications can make use of the system through an API
- Abstraction
  - Functionality is split hierarchically among the different layers: iRODS, mid-tier, front-end;
  - iRODS layer implements basic functionality and exposes it to higher layers through an API
- Containerization: Docker for services, VMs for iRODS/resource servers
- Robust: vertically and horizontally
RUG RDMS: Application design

NGINX

- Proxy
- Request forwarding
RUG RDMS: Application design

**NGINX**
- Proxy
- Request forwarding

**Mid-tier Web Service**
- Web Interface
- Projects management
- Users management
- Context management
- Data management

**Mid Tier DB**
- Projects/Users storage
- Roles
- Sessions storage

**Login Web Service + Redis DB**
- Login
- Cookies
- External accounts
RUG RDMS: Application design

NGINX
- Proxy
- Request forwarding

Mid-tier Web Service
- Web Interface
- Projects management
- Users management
- Context management
- Data management

Mid Tier DB
- Projects/Users storage
- Roles
- Sessions storage

Login Web Service + Redis DB
- Login
- Cookies
- External accounts

iRODS Web Service
- iRODS connection
- iRODS requests
- iRODS default search
RUG RDMS: Application design
RUG RDMS: Application design

- command line interface
- Web Dav
- other...
RUG RDMS: Application design

Metadata

- Mutable metadata
- Immutable metadata

- User generated metadata from Web interface
- User generated metadata from command line interface

- Automatic extracted metadata
- System metadata

System metadata is a number of metadata attributes in different formats that are used by the RDMS to control data workflows
- Mutable lists of metadata (id's)
- Object definition: project/dataset etc.
Web Interface
RUG RDMS: Web Interface

My folder
a.tsyganov@rug.nl

Filter by name..
# RUG RDMS: Web Interface

![Web Interface Screenshot](https://research.web.rug.nl/rdmswebapp/

## Main Menu
- **Data browser**
- **Groups and Team drives**
- **Projects**
- **Metadata Templates**
- **Contact support**
- **User profile**

## Data Browser

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Creator</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRTtest</td>
<td></td>
<td><a href="mailto:a.tsyganov@rug.nl">a.tsyganov@rug.nl</a></td>
<td>Oct, 22 2019</td>
</tr>
<tr>
<td>test</td>
<td></td>
<td>irodeo</td>
<td>Oct, 22 2019</td>
</tr>
<tr>
<td>UMCN_PILOTS_SHARE</td>
<td>2.1 MB</td>
<td><a href="mailto:a.tsyganov@rug.nl">a.tsyganov@rug.nl</a></td>
<td>Jun, 02 2021</td>
</tr>
<tr>
<td>stickers.jpg</td>
<td>200 B</td>
<td><a href="mailto:a.tsyganov@rug.nl">a.tsyganov@rug.nl</a></td>
<td>Oct, 29 2020</td>
</tr>
<tr>
<td>test.txt</td>
<td></td>
<td><a href="mailto:a.tsyganov@rug.nl">a.tsyganov@rug.nl</a></td>
<td>Oct, 29 2020</td>
</tr>
</tbody>
</table>
RUG RDMS: Web Interface

Objects tree

Files/Folders
RUG RDMS: Web Interface

Object information:
- permissions
- metadata
- audit
- iRODS metadata
- shared links
RUG RDMS: Web Interface

Background process to extract metadata
RUG RDMS: Web Interface

Extracted metadata
iRODS rules
RUG RDMS: iRODS rules

- Custom rules:
  - delayed rules and monitoring
  - copy data with permissions & metadata
  - automatic metadata extraction
  - custom policies engine
  - audit and permission control extra rules

- Rules to support tape library archiving
Custom policies engine
RUG RDMS: Custom policies engine

Data object: file or folder
RUG RDMS: Custom policies engine

Data object: file or folder

Special metadata as json

- Name:
  sysmdt_rdms_policy_<unique hash of the full path to object>
- Value:
  ```json
  {"policy_name" : <policy_name>,
   "policy_creator" : <policy_creator>,
   "input_parameters" : {
     <parameter_name>: <parameter_value>,
     <parameter_name>: <parameter_value>,
     <parameter_name>: <parameter_value>,
     ...
   }
  }
  ```
- Unit:
  POLICY|<policy type>|<policy status>|<policy user>
RUG RDMS: Custom policies engine

Data object: file or folder
/testZone/home/Projects/project0_5n1 => one direction hash transformation => 2c7197f0a89e1c842180756537534a81a069be79e8ec6fa1473af21c

Name: sysmdt_rdms_policy_2c7197f0a89e1c842180756537534a81a069be79e8ec6fa1473af21c

Value:
{"policy_name" : "project_user_participation_enddate",
"policy_creator" : "atsG",
"input_parameters" :
{
  "user_name" : "atsG",
  "end_date" : "10/06/2021 11:52",
  "date_format" : "%d/%m/%Y %H:%M"
}
}

Unit: POLICY|PROJECT|TORUN|atsG
RUG RDMS: Custom policies engine

Data object: file or folder

Policy metadata
RUG RDMS: Custom policies engine

Data object: file or folder

Mid-tier database

Policy metadata

{ key, unique_value_hash}
RUG RDMS: **Custom policies engine**

- **Data object:** file or folder
- **Policy metadata**
  - `{ key, unique_value_hash }

---

**Query:**

```
> ugrdmsdb=> \d RRREGISTERED_POLICIES_SYS
```

**Table: public.rrregistered_policies_sys**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_id</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>policy_verification_id</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>policy_status</td>
<td>character varying(255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>created</td>
<td>timestamp without time zone</td>
<td></td>
<td></td>
<td>now()</td>
</tr>
<tr>
<td>changed</td>
<td>timestamp without time zone</td>
<td></td>
<td></td>
<td>now()</td>
</tr>
<tr>
<td>policy_object</td>
<td>text</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Indexes:**

[Image references and university logos remain as is]
RUG RDMS: Custom policies engine

- Code injection protection: if verification is turned on then only policies with the correct verification hash will be triggered.
- Protection from a manual change of the metadata for policies
RUG RDMS: Custom policies engine

Server cron job (as rods user)
RUG RDMS: Custom policies engine

Server cron job (as rods user)

icommands - get all metadata that are policies to run
RUG RDMS: Custom policies engine

Server cron job (as rods user)

icommands - get all metadata that are policies to run

get policy information:
- metadata id
- user that needs to run policy
RUG RDMS: Custom policies engine

Server cron job (as rods user)

`icommands` - get all metadata that are policies to run

get policy information:
- metadata id
- user that needs to run policy

set irods environment username and run asynchronously `irules` with metadata id as input
**RUG RDMS:** Custom policies engine

Server cron job (as rods user)

- icommands - get all metadata that are policies to run

get policy information:
- metadata id
- user that needs to run policy

set irods environment username and run asynchronously irules with metadata id as input
RUG RDMS: Custom policies engine

## Server cron job (as rods user)

- icommands - get all metadata that are policies to run

## irule

- pass execution to the python irule
- get metadata key/value/unit by metadata id

## get policy information:

- metadata id
- user that needs to run policy

## set irods environment username and run asynchronously irules with metadata id as input
RUG RDMS: Custom policies engine

Server cron job (as rods user)

icommands - get all metadata that are policies to run

get policy information:
- metadata id
- user that needs to run policy

set irods environment username and run asynchronously irules with metadata id as input

irule

pass execution to the python irule

get metadata key/value/unit by metadata id

get python policy function name and parameters from the metadata value
RUG RDMS: Custom policies engine

Server cron job (as rods user)

icommands - get all metadata that are policies to run

get policy information:
- metadata id
- user that needs to run policy

set irods environment username and run asynchronously irules with metadata id as input

irule

pass execution to the python irule

generate metadata key/value/unit by metadata id

generate python policy function name and parameters from the metadata value

execute policy

update policy status
RUG RDMS: Custom policies engine

```python
from .policy_project_user_participation_enddate import run_policy_project_user_participation_enddate

CAVAILABLE_POLICIES = {
    "project_user_participation_enddate": run_policy_project_user_participation_enddate
}

# method to run policy that we've fetched from the metadata of the object
def run_policy(self):
    l_function_name, l_parameters = self.parce_policy()  # parce json
    if self.policyIsValid():
        if self.c_namespace.CAVAILABLE_POLICIES.has_key(l_function_name):
            self.c_namespace.CAVAILABLE_POLICIES[l_function_name](self, l_parameters)  # execute code
```
Metadata Templates
Why our approach?

• Metadata is the necessary component to transform data into knowledge.
• A well-written metadata template is crucial to tag the data being stored.
• Several requests for metadata template in different research domains:
  – social sciences, archeology, microbiology
There is no solution of *one-size-fits-all*

- Each **research domain** has **specific** metadata-attributes
- Within the same domain **research questions** may have different collection of tags
- The technical skills of the users are very different

Provide the *User* with the possibility to *define* domain/research **specific** metadata templates
RUG RDMS: Metadata Templates - Technical solution

- JSON Schema
- Template components Tree
- XSD - schema
- Validation
- Dynamically created form
- Metadata AVUs
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Base type</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>Enter description</td>
<td>string</td>
</tr>
<tr>
<td>Camera</td>
<td>Enter description</td>
<td>string</td>
</tr>
<tr>
<td>Angle</td>
<td>Enter description</td>
<td>Azimuth Angle</td>
</tr>
<tr>
<td>Subject</td>
<td>Enter description</td>
<td>string</td>
</tr>
<tr>
<td>Number of images</td>
<td>Enter description</td>
<td>integer_between_10_and_20</td>
</tr>
</tbody>
</table>
Self defined structure

Attributes are validated at input
Auditing
RUG RDMS: Auditing

RDMS

RDMS

iRODS Audit Plugin

RabbitMQ

logstash

elastic

kibana

RDMS midtier
"pep_regex_to_match" to reduce the amount of data
RUG RDMS: Auditing

“pep_regex_to_match" to reduce the amount of data

Ruby script to remove empty fields in documents
“pep_regex_to_match" to reduce the amount of data.

Ruby script to remove empty fields in documents.

5% of the original amount of data.
**Index Lifecycle Management**

- Actively queried
- Written Data
- Required phase
Index Lifecycle Management

- Actively queried
- Written Data
- Required phase

Hot ➔ Warm ➔ Cold

- Actively queried
- No data written
- Moved to less performant hardware
RUG RDMS: Elastic Search - Index Lifecycle Management

- Actively queried
- Written Data
- Required phase

- Less frequently queried data
- Searchable
- Before deleted, snapshot is created and stored to long term storage

Index Lifecycle Management

- Actively queried
- No data written
- Moved to less performant hardware
RUG RDMS: What's next?

- Make code open source & improve deployment
- Integrate switch between different iRODS zones
- Finish customers use cases (currently there are 7 major project running from different faculties)
- Make iRODS NFS mount
- File level versioning
- Multi factor authentication & External users
- Fail2ban
- Windows ingest large files
- Add metadata via mount
- Next version of the metadata templates
- Auditing - SLM snapshots