LUG 25
Robinhood 4: the policy engine toolkit

Guillaume Courrier <guillaume.courrier@cea.fr>

A little bit of history

- 1999: first policy engine at CEA to run purge policies
- 2005: Robinhood v1
  - Metadata in memory: limit scalability
  - Purge policies, OST aware, multithreaded
- 2009: open source
- 2009: Robinhood v2
  - Introduction of a SQL database to increase scalability
  - More complex policies
- 2015: Robinhood v3
  - More modular: possibility to create custom policies
- 2018: Robinhood v4

Robinhood 3 Update

- Version 3.2 released this year: https://github.com/cea-hpc/robinhood/releases/tag/3.2.0
  - Support for project ID
  - policy sort order by size, e.g. lru_sort_attr = size;
  - asc/desc modifiers for sort order, e.g. lru_sort_attr = size(desc);
  - policy trigger thresholds as percentage of available inodes: high/low_threshold_cntpct = xx%;
  - Better Lustre 2.15 support
  - RHEL9.4 OS family support

Why Robinhood 4?

- Integration of more scalable databases (e.g. MongoDB)
- More modular:
  - Each functionalities of a policy engine are implemented by a single command
  - Minimal configuration
  - Database and file system specific code is clearly isolated and relatively small
- Support for modern Lustre features:
  - FLR, DOM, DNE v2/v3…

But the main architecture remains the same:

- Mirror all the metadata in an external database and run policies based on the mirror’s content

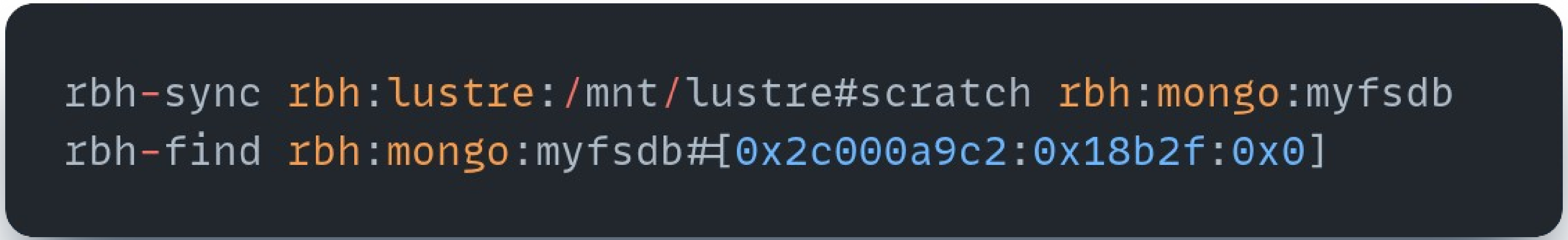
1. Core concepts

FS Entries, Backends and URI

FS Entries, Backends and URI

- An FS entry is essentially an inode
- A backend contains FS entries and lets us read or update them
  - File System backends: POSIX, Lustre, MPIFileUtils
  - Object Store: HESTIA (Object store developed during the IO-SEA Euro HPC Project)
  - Database: Mongo DB, SQLite (in progress), MPIFileUtils

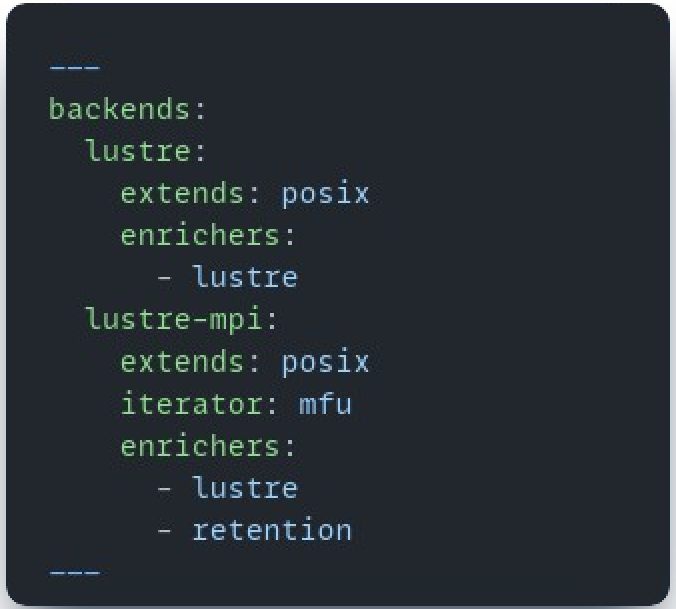
- Backends and FS entries are identified by a URI:



```
rbh-sync rbh:lustre:/mnt/lustre#scratch rbh:mongo:myfsdb
rbh-find rbh:mongo:myfsdb#[0x2c000a9c2:0x18b2f:0x0]
```

POSIX backend and extensions

- The POSIX backend will fetch all the regular metadata (stat and xattrs)
- Extensions can be added to fetch additional info
  - Currently two extensions: Lustre and retention
  - New extensions can be added



```yaml
---
backends:
  lustre:
    extends: posix
    enrichers:
      - lustre
  lustre-mpi:
    extends: posix
    iterator: mfu
    enrichers:
      - lustre
      - retention
---
```

2. Robinhood in action

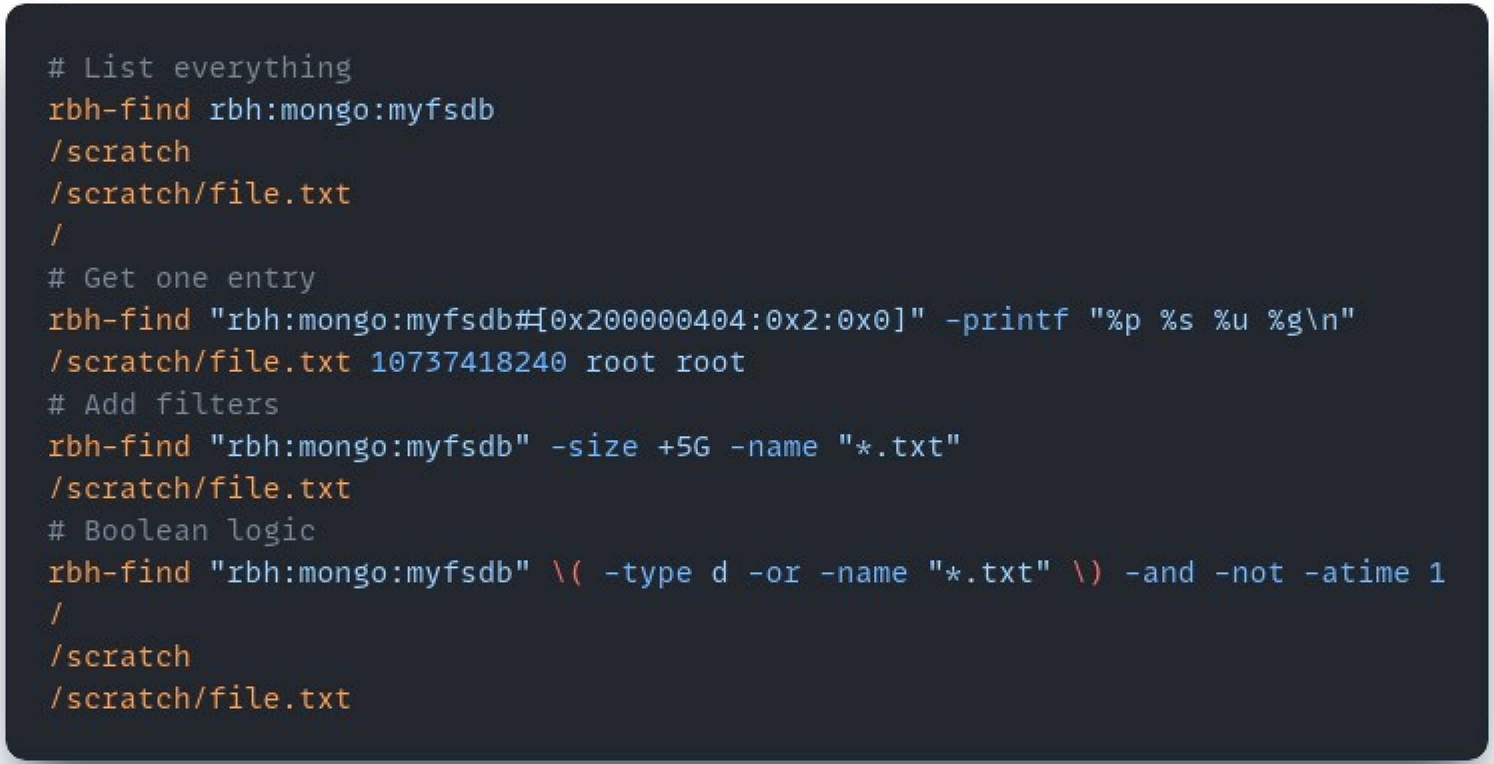
Synchronization, queries, reports

Synchronizing backends

```
# Scan the whole FS
rbh-sync rbh:lustre:/mnt/myfs rbh:mongo:myfsdb
# Scan subdir "scratch" (branching)
rbh-sync rbh:lustre:/mnt/myfs#scratch rbh:mongo:myfsdb
# Scan one entry
rbh-sync --one rbh:lustre:/mnt/myfs#scratch/file.txt rbh:mongo:myfsdb
```

Querying backends

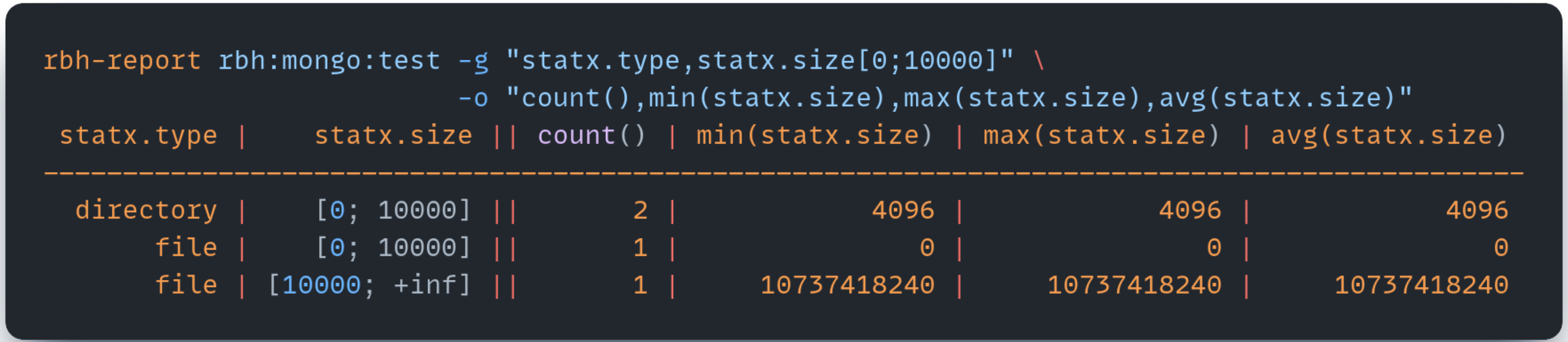
- rbh-find: implements most of the features of the regular find
- rbh-lfind: Lustre aware version (support of –ost-index, -mdt-index, -pool, -hsm-state…)



```
# List everything
rbh-find rbh:mongo:myfsdb
/scratch
/scratch/file.txt
/
# Get one entry
rbh-find "rbh:mongo:myfsdb#[0x200000404:0x2:0x0]" -printf "%p %s %u %g\n"
/scratch/file.txt 10737418240 root root
# Add filters
rbh-find "rbh:mongo:myfsdb" -size +5G -name "*.txt"
/scratch/file.txt
# Boolean logic
rbh-find "rbh:mongo:myfsdb" \( -type d -or -name "*.txt" \) -and -not -atime 1
/
/scratch
/scratch/file.txt
```

Reports (v1)

- Generic implementation that can create various kinds of reports
  - --group-by: specify on which data to aggregate results
  - --output: specify what to compute in the report
- V1 computes everything on each invocation of the command
  - Future versions will precompute parts of those information to increase performance of common reports



```
rbh-report rbh:mongo:test -g "statx.type,statx.size[0;10000]" \
                          -o "count(),min(statx.size),max(statx.size),avg(statx.size)"
 statx.type |    statx.size || count() | min(statx.size) | max(statx.size) | avg(statx.size)
-----------------------------------------------------------------------------------------------
  directory |     [0; 10000] ||       2 |            4096 |            4096 |            4096
       file |     [0; 10000] ||       1 |               0 |               0 |               0
       file | [10000; +inf] ||       1 |     10737418240 |     10737418240 |     10737418240
```

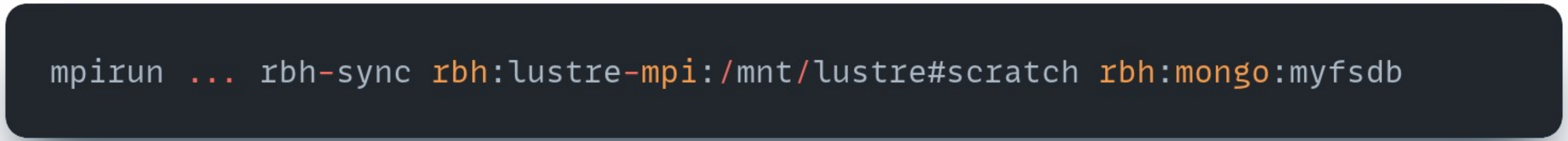
Updating the backend with changelogs

```
# Create a few changelogs
mkdir lug25
touch lug25/slides.tex
lfs getstripe -m lug25
1
lfs getstripe -m lug25/slides.tex
1
lfs changelog myfs-MDT0001 cl1
1 02MKDIR 13:11:15.725326452 2025.03.25 0x0 t=[0x240000404:0x1:0x0] ...
2 01CREAT 13:11:52.589025431 2025.03.25 0x0 t=[0x240000404:0x2:0x0] ...
3 11CLOSE 13:11:52.608211252 2025.03.25 0x42 t=[0x240000404:0x2:0x0] ...

# Process changelogs
rbh-fsevents -e rbh:lustre:/mnt/myfs src:lustre:myfs-MDT0001?ack-user=cl1 rbh:mongo:test
# The entry has been added
rbh-find rbh:mongo:test -name "*.tex"
/lug25/slides.tex
```

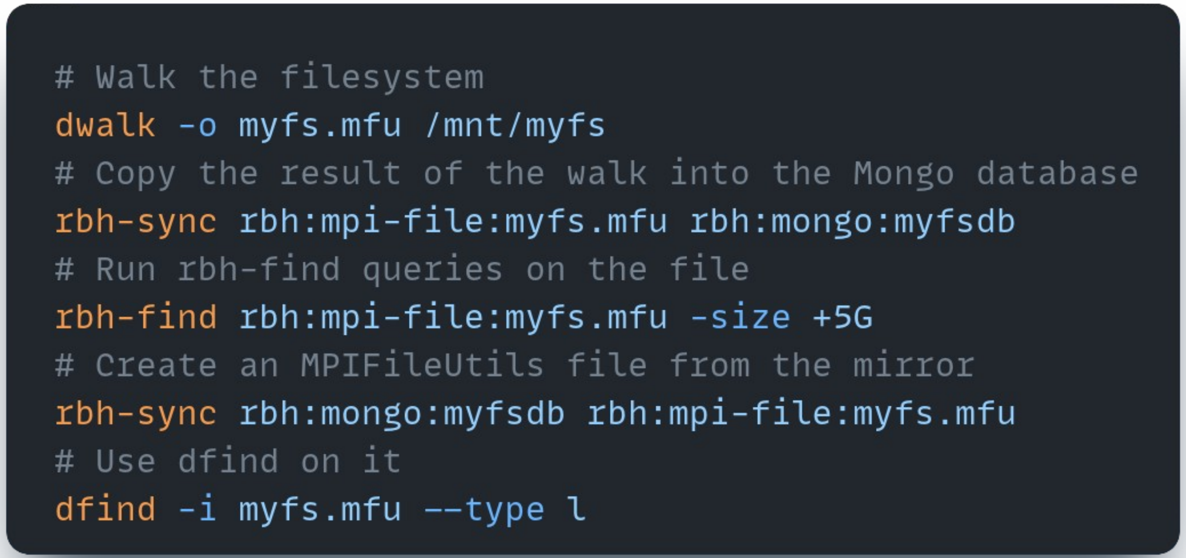
MPIFileUtils

- Distributed scan using the API of MPIFileUtils:



```
mpirun ... rbh-sync rbh:lustre-mpi:/mnt/lustre#scratch rbh:mongo:myfsdb
```

- Interoperability with MPIFileUtils tools:



```
# Walk the filesystem
dwalk -o myfs.mfu /mnt/myfs
# Copy the result of the walk into the Mongo database
rbh-sync rbh:mpi-file:myfs.mfu rbh:mongo:myfsdb
# Run rbh-find queries on the file
rbh-find rbh:mpi-file:myfs.mfu -size +5G
# Create an MPIFileUtils file from the mirror
rbh-sync rbh:mongo:myfsdb rbh:mpi-file:myfs.mfu
# Use dfind on it
dfind -i myfs.mfu --type l
```

Retention

- Users can set an expiration date on their files and/or directories using an extended attribute
- Robinhood can then list expired entries and perform actions on them (purge, archive…)

```
# User sets the expiration attribute
setfattr -n user.expires -v "$(date -d "yesterday" +%s)" /mnt/myfs/scratch/file.txt
# rbh-sync or rbh-fsevents...

date
Tue Mar 25 13:30:01 UTC 2025
# We can display the expiration date using rbh-lfind (not rbh-find)
rbh-lfind rbh:mongo:test -printf "%p %E\n"
/scratch/file.txt Mon Mar 24 13:28:58 2025
# Or list expired files
rbh-lfind rbh:mongo:test -expired
/scratch/file.txt
# And even remove it
rbh-lfind rbh:mongo:test -expired -delete
stat scratch/file.txt
stat: cannot statx 'scratch/file.txt': No such file or directory
```

Future work

- First deployments this year
  - Optimizations, benchmarks
- rbh-undelete: recreate a file removed from Lustre using the info from the DB and the HSM archive for content
- Scan garbage collection: remove stale entries from a backend
- Design and first implementation of the Policy engine

Thank you, any questions?

Source code:

- https://github.com/robinhood-suite/robinhood4

Submitting patches:

- https://review.gerrithub.io/q/project:robinhood-suite/robinhood4+status:open