FROM RESEARCH TO INDUSTRY

RobinHood Project Update

Robinhood User Group 2017

Thomas Leibovici <thomas.leibovici@cea.fr>

October 3rd, 2017

www.cea.fr

Topics

- Project update
  - Releases & stats
  - Community resources
- What's new in Robinhood?
  - New features in robinhood 3.1
  - RUG'16 issues addressed in robinhood 3.1
- What's next?
  - Next development plans

Project update

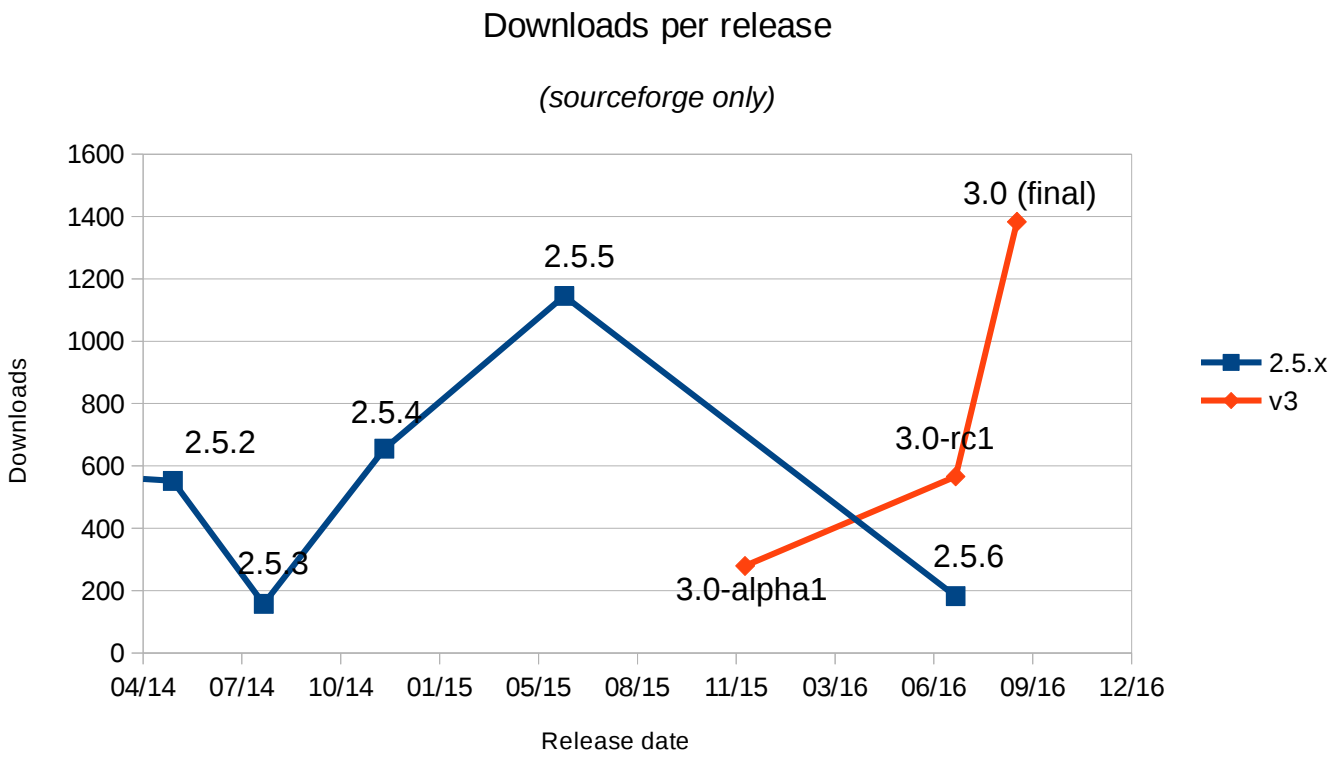
Latest Releases

- Robinhood 3.0 : Sept 2016
  - Support:
    - RHEL 6, 7
    - Validated on Lustre 2.1 to 2.8
- Robinhood 3.1 : Sept 2017
  - Support:
    - RHEL 6, 7
    - Validated on Lustre 2.1, 2.4, 2.5, 2.7, 2.8, 2.9, 2.10 (no PFL support for now)
- No minor releases
  - Some vendors maintain specific versions for their clients
  - Get git “master” branch to get the latest fixes

```
git clone https://github.com/cea-hpc/robinhood.git
```

Release Stats

- The number of users is still growing
  - 2.5.5 vs 3.0: 1150 → 1400 downloads



Community Resources

- Github: **cea-hpc/robinhood**
  - Git repository: https://github.com/cea-hpc/robinhood.git
  - Documentation in the wiki
  - Issue reporting/tracking

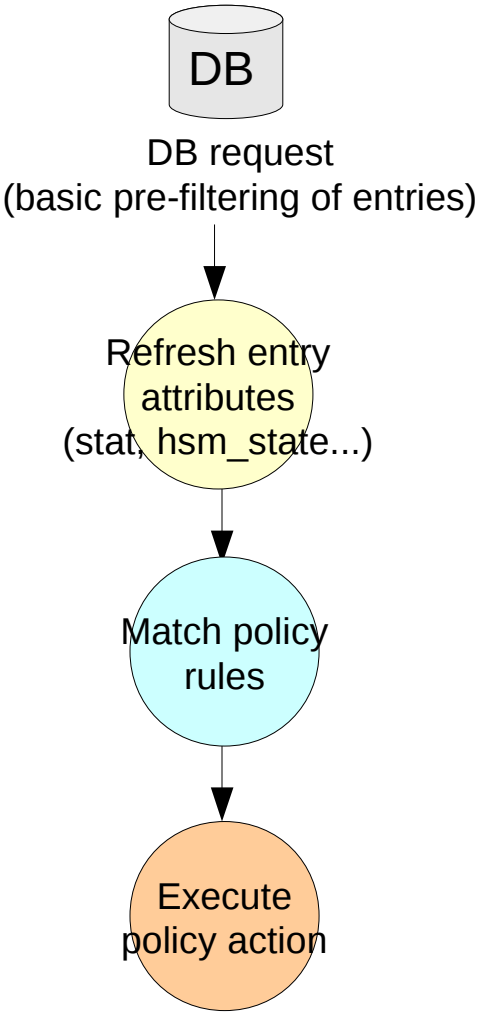
- Gerrithub (code review):
  - https://review.gerrithub.io
  - Project: cea-hpc/robinhood
  - All landings goes through it (please, no “pull requests” on github)
  - Bound to an automated test system hosted at CEA (jenkins)

- Managed by sourceforge:
  - Mailing lists:
    - robinhood-news@lists.sf.net
    - robinhood-support@lists.sf.net
    - robinhood-devel@lists.sf.net
  - Download center

What's new in Robinhood 3.1?

Policy workflow until robinhood 3.0

V3.0 and before



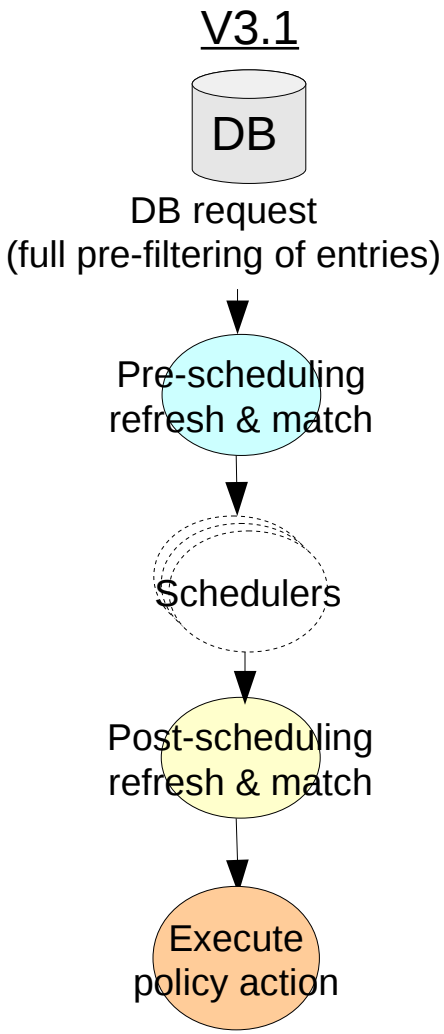
Issues:

- Filtering too weak: many returned entries won't match the rules
  → causes useless attribute refresh & rule matching
- Expensive requests: jerky workflow when using 'max_action_count'

- Bunches of FS operations to refresh entry attributes.
  → Sometimes useless if policy rules don't need fresh attributes
  → Slows down policy runs

- Exploding wait queue of systems with asynchronous actions (in particular, Lustre/HSM action queue)
- Sub-optimal mix of operations (small vs. big files)

Enhanced policy workflow (robinhood 3.1)



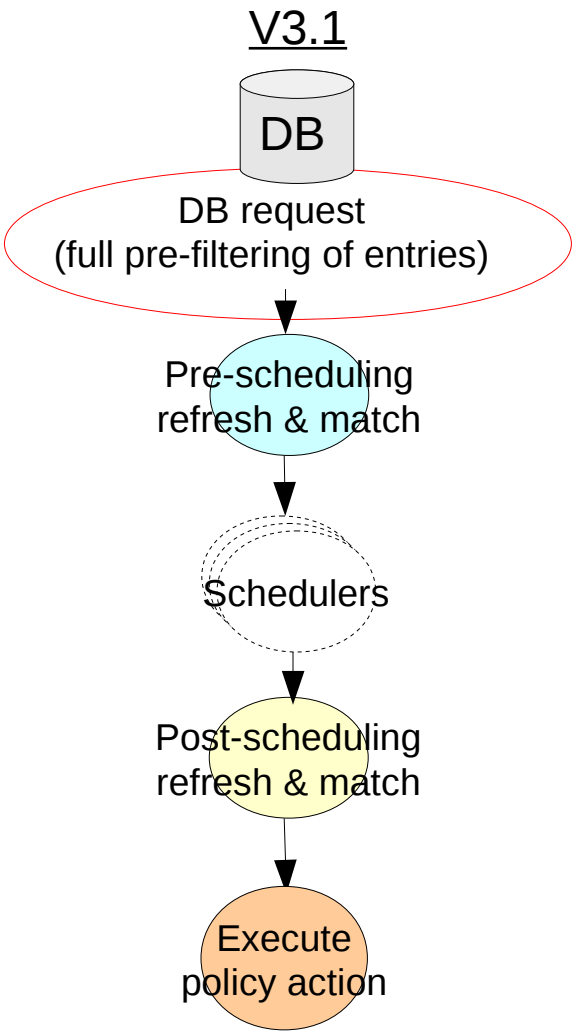
Enhancements:

- Full conversion of policy rules to DB request
→ minimizes entries to be processed

- Smarter and configurable matching behavior (before and after scheduling)

- Schedulers :
  - Can delay, reorder, skip entries...
  - Plugins (you can implement your own)
  - Stackable
  - Provided implementation: “common.rate_limit”
    - Allow limiting the rate of actions (count and/or size)

Enhanced policy workflow (robinhood 3.1)



Enhancements:

- **Full conversion of policy rules to DB request**
  → minimizes entries to be processed

- Smarter and configurable matching behavior (before and after scheduling)

- Schedulers :
  - Can delay, reorder, skip entries...
  - Plugins (you can implement your own)
  - Stackable
  - Provided implementation: “common.rate_limit”
    - Allow limiting the rate of actions (count and/or size)

Fully converting policy rules to SQL (1/3)

Improve pre-filtering of entries in policy runs

- Cray contribution (Davide Tacchella)
- What rules were converted to SQL:

Before

```
ignore_fileclass = foo;
ignore_fileclass = moo;
ignore { last_mod < 1h }
ignore { last_access < 1h }

rule single_rule {
    target_fileclass = bar;
    condition { last_access > 1d }
}
```

- Simple ignore statements (fileclass, or single criteria)
- Simple rule conditions, if there is only 1 rule

- No DB pre-filtering for other cases:
  - More entries are retrieved from the DB and matched in the policy run itself
  - => longer policy runs

Fully converting policy rules to SQL (2/3)

- All policy rules based on DB fields now converted to SQL:

<u>In robinhood 3.1</u>

```
ignore_fileclass = foo1;
ignore_fileclass = moo;
ignore { last_mod < 1h and
          (owner == root or name == “save*”) }

rule rule1 {
   target_fileclass = foo2;
   target_fileclass = foo3;
   condition { last_access > 1d
               or (name == *.log and last_mod > 6h) }
}

rule rule2 {
   target_fileclass = boo;
   condition { last_mod > 1h }
}

...
```

- All ignore statements (even with nested conditions)
- All rules even with nested conditions
- **Results in faster policy runs**

Policy run performance

Example of “alert” policy

- Applying an “alert” policy on half a billion entries filesystem, with 1000 entries matching alert rules
  - Alert rules:
    - large files > 500GB
    - directories with more than 50k entries
  - Robinhood 3.0: 1~6 hours
  - Robinhood 3.1: less than 30 sec
- Explanation:
  - V3.0: millions of entries returned by the SQL request.
    Update of entry attributes + rule matching for all these entries.
  - V3.1: SQL request only select entries matching alert rules.
    Update of attributes only for those matching entries.

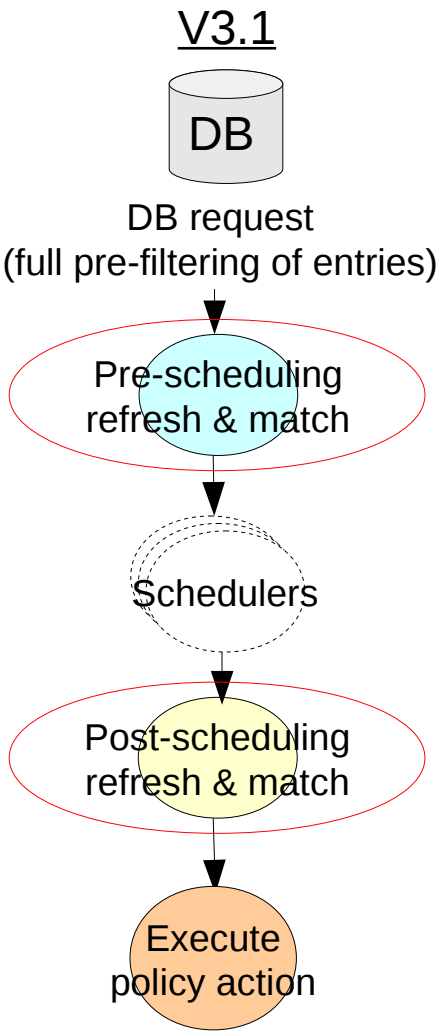
Fully converting policy rules to SQL (3/3)

Drawback and workarounds

- If the contents of the DB is outdated, policy run may not consider some entries that now match the policy rules
  - Delay in changelog processing
  - If your change fileclass definitions
- To force rematching entries, set in the policy configuration:
  - `recheck_ignored_entries = yes`
  - This disables the pre-filtering from DB and force rematching policy rules
- Or you can define an 'update' policy to update DB contents in background:

```
define_policy update {
    default_action = none; # simply update entries, run no action
    scope { type == file }
    status_manager = none;
    default_lru_sort_attr = none;
}
```

Enhanced policy workflow (robinhood 3.1)



Enhancements:

- Full conversion of policy rules to DB request
→ minimizes entries to be processed

- **Smarter and configurable matching behavior (before and after scheduling)**

- Schedulers :
  - Can delay, reorder, skip entries...
  - Plugins (you can implement your own)
  - Stackable
  - Provided implementation: “common.rate_limit”
    - Allow limiting the rate of actions (count and/or size)

Customizable attribute update for matching

- When using schedulers, rule matching is done twice:
  - A first matching is done before the scheduling to avoid filling scheduler queues with uninteresting entries
  - A second time when the action is actually scheduled, to check if the entry still matches policy rules

- The matching behavior and the tested attributes in these 2 steps is configurable:

  - **NONE**: no matching is done (pass through)
  - **CACHE_ONLY**: the matching is done using attributes from DB (no FS request)
  - **AUTO_UPDATE**: only attributes needed by the specified policy rules are refreshed
  - **FORCE_UPDATE**: force updating entries attributes before matching

- By default:
  - Matching before scheduling is based on “cached” attributes
  - Matching after scheduling is “auto”

How it looks in the configuration

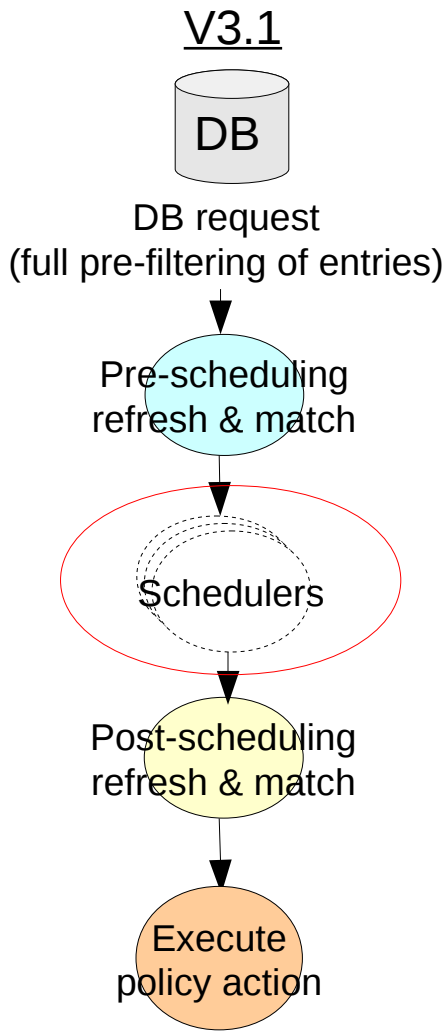
```
<policy>_parameters {
        pre_sched_match = cache_only;
        post_sched_match = auto_update;

        schedulers = common.rate_limit;

        rate_limit {
                period_ms = 100;
                max_count = 100; # 100/100ms = 1k/sec
                max_size = 1GB;  # 1GB/100ms = 10GB/sec
        }
}
```

- 2 new parameters to control attribute refresh & matching
  - `pre_sched_match, post_sched_match`
- 1 new parameter : “schedulers”
  - Coma-delimited list of schedulers (stackable)
- 1 configuration block per scheduler

Enhanced policy workflow (robinhood 3.1)



Enhancements:

- Full conversion of policy rules to DB request
→ minimizes entries to be processed

- Smarter and configurable matching behavior (before and after scheduling)

- **Schedulers :**
  - **Can delay, reorder, skip entries...**
  - **Plugins (you can implement your own)**
  - **Stackable**
  - **Provided implementation: “common.rate_limit”**
    - Allow limiting the rate of actions (count and/or size)

Schedulers: implement your own

- Scheduler are managed as plugins
  - Provided with robinhood: **`common.rate_limit, common.max_per_run`**
  - You can implement your own: `myplugin.sched1, myplugin.sched2, ...`
- A scheduler provides a function called by robinhood with a callback

```
sched_schedule(<entry info>, callback_func, callback_param);
```

- The scheduler can then queue the entry internally
- When it decides to schedule the action, the scheduler calls the callback function for the entry
- The scheduler can also notify robinhood to:
  - suspend entry scheduling for a while (e.g. if its internal queues are full)
  - skip a given entry
  - stop current run after finishing queued actions
  - stop current run immediately
- A scheduler must also provide functions to manage its internal state:
  - load configuration, initialize, reset...

Pre/post policy run commands

Running commands before/after policy runs

- Simple feature, but useful
- Example use-case:
  - Before run: Create empty list file
  - Policy run: add path of old files to the list
  - After run: Send list of old files to users
- Specified in `<policy>_parameters`:
  - `pre_run_command = “my_start_script.sh {fsname}”;`
  - `post_run_command = “my_end_script.sh {fsname}”;`

New policy plugin: modeguard

Modeguard status manager

- Contribution of Stanford University (Stephan Thiell)
- Check access rights of entries matching policy scope → Audit, report
- Modeguard policies can:
  - Force setting permission bits of some entries → Enforce
    - e.g. set gid bit for some directories
  - Force clearing permission bits of some entries
    - e.g. clear 'w' flag for other
- Configuration example:

```
%include "includes/modeguard.inc"
modeguard_config {
        # set_mask = 2000;
        clear_mask = 0002;
}
# clear 'w' flags of user's directories
modeguard_rules {
        rule secure_user_dir {
                fileclass = user_dirs;
                condition { modeguard.status != ok }
        }
}
```

Example policies

More examples in robinhood v3.1

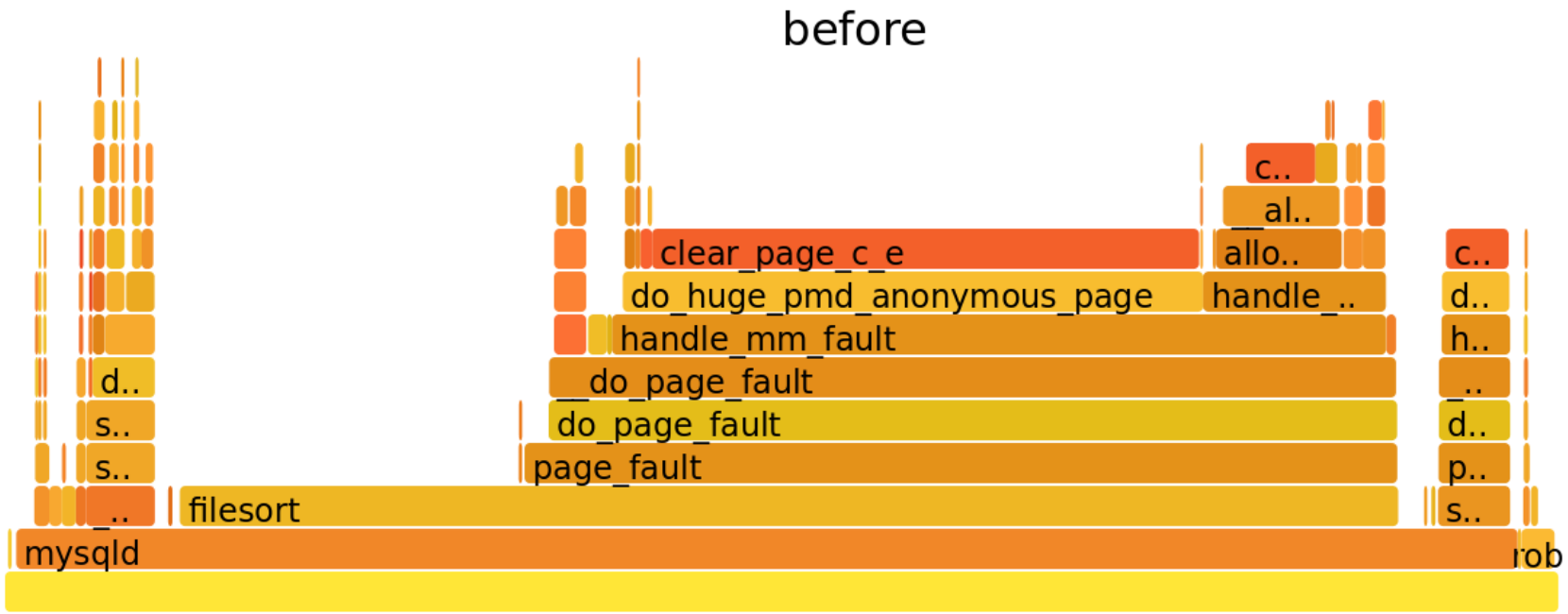
- Examples installed in `/etc/robinhood.d/templates`:

```
example_alerts.conf
example_checksum.conf
example_cleanup.conf
example_lhsm.conf
example_modeguard.conf
example_rmdir.conf
```

- You can use them as is, or use them as examples to write your own
- Note: these config files can be merged if you need to run several of these policies in a single robinhood instance

Ingest rate optimization (1/2)

Profile when processing changelogs, or scanning



- Most of the time spent in mysqld in filesort()
- CPU load: ~450% mysqld, ~5-10% robinhood
- Due to a request that sorted a list of 1 entry most of the time...
  (entry path ordering, most recent first)
- Resulted in calling this expensive function in mysqld

Ingest rate optimization (2/2)

After modifying the request

- Less time spent in mysqld, different call profile
- CPU load: mysqld 300%, robinhood 100%
- Robinhood processing speed : about x7 !
- Together with other v3.1 optimizations: about x10

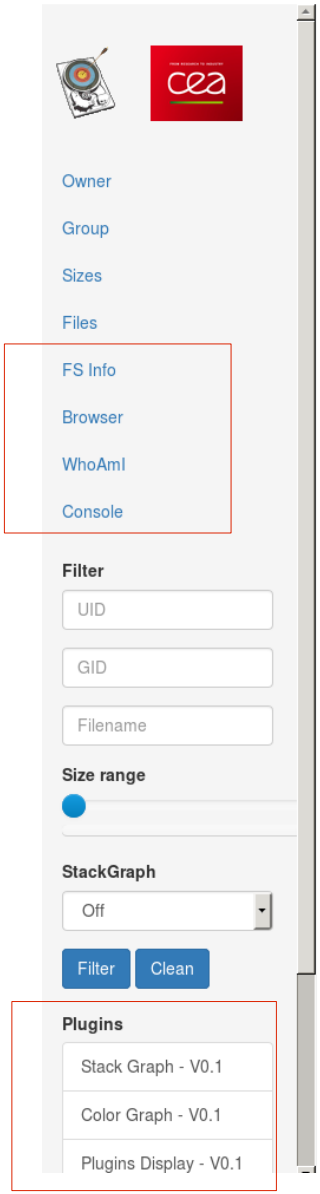
REST API & Web UI enhancements

- All robinhood info exposed through the REST API
  - Make it possible to replace parsing of rbh-report output by a stable API
  - Can be queried remotely with a simple HTTP client
  - Fine-grained access control (by user, groups, ...)
- Plugin mecanism for REST or Web interface
  - Make it possible to extend the REST API
  - Make it possible to add custom charts and reports to web interface
  - New sections provided as plugins: e.g. namespace browsing, internal stats...
  - Other enhancements

WebGUI plugins: overview

New sections

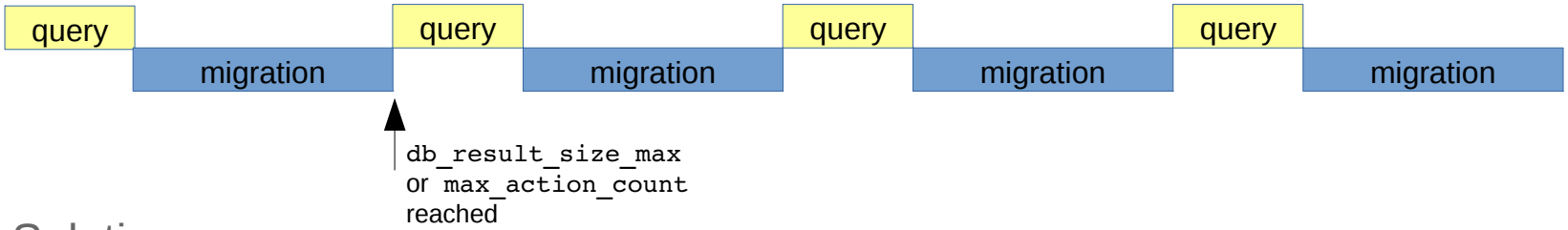
Loaded plugins



Issues exposed at RUG 2016 addressed in robinhood 3.1

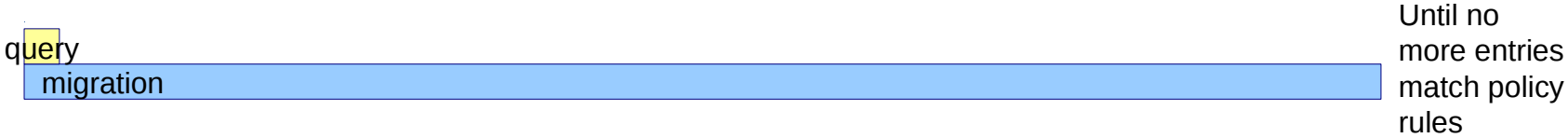
RUG'16 issues: Lustre/HSM archive

- Stanford University: *"max_action_count" has been very useful to avoid too many Lustre/HSM actions*
- Cray: *no overlap of query/migration*



- <u>Solutions:</u>
  - [v3.1] With "rate_limit" scheduler, it is no longer needed to abort policy runs after a given number of actions. Robinhood just makes 1 query and then emit requests smoothly without saturating the HSM action queue
  - Increase `db_result_size_max` (e.g. 1M)
  - Use "`lru_sort_attr = none`" to make the query instantaneous

- <u>Result:</u>



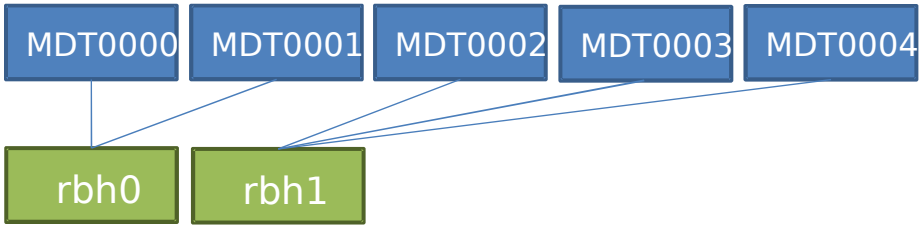
RUG'16 issues: maximize bandwdith and op/sec

- Stanford University: *Would love an "interleaved archiving mode" to mix smallfiles and bigfiles*
  - *ideally by percent of each (eg. 10% smallfiles, 90% bigfiles)*
  - *to push smallfiles while bigfiles are transferring, thus maximizing both transfer bandwidth and max QPS the cloud provider allows*

- <u>Solution:</u>
  - [v3.1] This can now be implemented as an action scheduler plugin
  - It could maintain 2 internal queues: 1 for small files, 1 for big files and schedule actions to maintain both high bandwidth and high op/sec

RUG'16 issues: namespace partitioning

- DKRZ:
  - Namespace split into multiple robinhood instances
  - Several MDTs per instance
  - *“Automatic generation of ignore list in robinhood configuration”*



- Robinhood 3.1 new feature:
  - Can restrict namespace scan to a set of directories (`scan_only` directive)

Instance 1

```
fs_scan {
    scan_only = /scratch/project1;
    scan_only = /scratch/project3;
}
```

Instance 2

```
fs_scan {
    scan_only = /scratch/project2;
    scan_only = /scratch/project4;
}
```

  - Much more convenient to add a new directory created with “lfs mkdir” rather than ignoring all other directories

RUG'16 wishes: query API

- Stanford University: *We heavily parse rbh-report, a API would be convenient for many scripts*

- Solution:
  - (v3.1) All information provided by 'rbh-report', 'rbh-find', 'rbh-du' are now available through the REST API

Work in progress and future plans

PFL support

PFL support implies deep changes in robinhood

- Change the way stripe information is stored
  - Database schema before PFL: 1 stripe info per file + list of stripes
  - With PFL: 1 stripe info + list of stripes for each file region
- Impact on policy rules:
  - Criteria before PFL:
    `ost_pool == "foo"` means the file is associated to pool "foo"
  - What does that mean with PFL?
  - Option 1: at least one region of the file is stored on pool "foo"
  - Option 2: change from file-level criteria to region-level criteria
    - Implies applying policies to file regions instead of whole files
- Refreshing of stripe info:
  - Based on "layout_gen" => in which PFL cases it changes?
  - Generation number for regions? Layout swap of regions? HSM flags...

New Database Layering

- Goals:
  - Support multiple types of databases, even NOSQL
    - Candidates: MongoDB, PostgreSQL
  - Leverage // databases
    - Unleash robinhood scalability
  - More flexible DB schema
    - Flexible entry id (not only 'fid' or 'inum'), useful for object stores
    - Flexible attribute set (custom policies, object stores)
    - PFL requirement (compound layout structures)

- Status:
  - Design in progress

Asynchronous accounting

- Goals:
  - Reduce the impact of aggregated stats updated on-the-fly (e.g. total volume per user…) on performance
  - Make it possible to distribute this processing on other servers
  - Make it possible to add new aggregated stats (e.g. per sub-tree, per process, changelog stats per user, per job, …) at will without impacting robinhood performance and scalability

- Status:
  - Dependant of database mechanisms (triggers, ...)
  - Impact of database re-layering
  - => integrated to new DB design

Other TODOs

- Improve performance of GC at end of scan
- Undelete enhancements
  - non only archived files, but also directories, symlinks...
- rbh-find: support more options (e.g. -perm) and more complex combinations of options (with -and, -or, parenthesis...)
- DNE: save MDT stripe info in DB
  - Can be useful for undelete cases
  - Can enable new features of reporting...
- DNE phase 2: support inode relocation (MIGRTD changelog)
- New WebUI features: quota visualisation, nagios plugin, ...

- Longer term: redesign the event processing to distribute it to multiple agents on multiple hosts.

Thank you for your attention !

Questions ?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA / DAM Ile-de-France | Bruyères-le-Châtel - 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00

DAM Île-de-France

Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019