RobinHood v3 and Beyond

LAD'15

Thomas Leibovici <thomas.leibovici@cea.fr>

SEPTEMBER, 23rd 2015
Motivations

Address more use cases

- Scheduling "migration" and "purge" is nice, but I'd like do other things on my filesystem!
  - Implement a trash mecanism
  - Rebalance OSTs
  - Data migration between pools
  - ...

- Allow defining custom policies for site-specific use-cases:
  - Data integrity checks
  - Data post-processing
  - ...

- Be able to manage all of this in a single robinhood instance

Goal: make robinhood even more generic and flexible
Motivations

Interacting with site environment
- Allow interactions with external components
  - Job scheduler
  - Monitoring infrastructure
  - HSM backends
  - ...
- As input, to influence policy decisions
- As output: actions and reports

Goal: make robinhood more modular
- allow the integration of vendor-specific and site-specific modules
Motivations

Make sysadmin life better
- Eliminate the need for writing scripts to apply massive actions
- Provide new reports about filesystem activity
- Improve unhandy features in v2.x
- Still improve robustness

Get ready for next generations of systems
- Scalability
- Manage heterogeneous filesystems (SSDs + disks + ...)
- Adapt to new storage paradigms
  - Object stores
Robinhood V3 Release Cycle

V3.X Release Cycle

- Several major changes are planned in the next couple of years
  - Turn Robinhood to an Exascale-class software
- V3.0 is a first step, but not the last stop
- V3.1, V3.2, … to follow quickly

Roadmap overview

- V3.0: Policy framework with plugin-based architecture
  - Generic policies
- V3.1: Enriched plugin ecosystem + enhanced workflows
- V3.X: Performance & scalability improvements
  - Horizontal scalability
  - Take benefit of new Lustre features
  - Support new storage systems
Robinhood v3 Plugin-Based Architecture

Robinhood core made generic

- Purpose-specific code moved out of robinhood core: now dynamic plugins loaded at run-time
- All policy behaviors made configurable
- Vendors/users can write their own plugins for specific needs
Plugins types (V3.0)

Status managers

- Manage specific state machines
- Provide specific callbacks for Changelogs, policy actions...
- Can possibly interact with external components
- Status managers for common use-cases shipped with robinhood
  - e.g. lhsm
- Vendors/users can provide their own implementations as separate plugins

Configurable actions

- Implementation as a plugins avoid forking external commands
- Possible calls to external APIs
- Easy specification in configuration file
  - e.g. `action = common.unlink`;
- Shipped with robinhood v3.0:
  - `common`: common filesystem actions (copy, unlink, move, ...)
  - `lhsm`: Lustre/HSM specific actions (archive, release, ...)
More plugin types in next versions

- **Triggers**: control policy application
  - When it starts, targeted entries, how much, …
  - e.g. (current) start a policy run on files of an OST when it gets full
  - e.g. (future) start a policy run on a given directory when a job ends

- **Custom attributes**
  - Allow maintaining custom attributes for entries in robinhood database
    - e.g. checksum, project, …
  - Robinhood framework to provide reporting features for them

- **DB connectors**
  - Allow using various DB backends for robinhood
    ...including **parallel databases**
  - The way to **scalability**!
Generic Policies (V3.0): Motivation

Before v3

- Static set of policies, statically defined
- 1 mode = 1 robinhood instance = 1 set of commands
- Instances can't coexist on the same filesystem

<table>
<thead>
<tr>
<th>Package</th>
<th>&quot;migration&quot; policy</th>
<th>&quot;purge&quot; policy</th>
<th>&quot;hsm_remove&quot; policy</th>
<th>&quot;rmdir&quot; policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>robinhood-tmpfs</td>
<td>-</td>
<td>rm (old files)</td>
<td>-</td>
<td>rmdir, rm –rf</td>
</tr>
<tr>
<td>robinhood-backup</td>
<td>Copy to storage backend</td>
<td>-</td>
<td>rm in storage backend</td>
<td>-</td>
</tr>
<tr>
<td>robinhood-lhsm</td>
<td>Lustre HSM archive</td>
<td>Lustre HSM release</td>
<td>Lustre HSM remove</td>
<td>-</td>
</tr>
</tbody>
</table>

Robinhood v2.x packages and policies

- E.g. Lustre/HSM purpose:
  - Package: robinhood-lhsm
  - Commands: rbh-lhsm-*
  - Only implements HSM-related policies (*archive, release, remove*)
  - Cannot manage other actions (delete old files, …)
Generic Policies (V3.0): Overview

Robinhood v3 generic policies

- 1 single Robinhood instance for all purposes:

<table>
<thead>
<tr>
<th>Package</th>
<th>Generic policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>robinhood</td>
<td>Fully configurable</td>
</tr>
</tbody>
</table>

- Robinhood core: **generic** policy implementation

- Specific aspects:
  - Specified by **configuration** (policy templates)
  - Possibly as specific **plugins** (dynamic libraries)

- Policies at will
  - Schedule all you want!
  - Just by writing a few lines of configuration
Example: configurable pool migration with just a few lines of config

 Declare the policy:

```plaintext
declare_policy move_pool {
  scope { type == file and status != ok }
  default_action = cmd("lfs migrate -p {pool} -c {count}");
  status_manager = basic; # ok/failed
}
```

 Specify rules:

```plaintext
move_pool_rules {
  rule migr_movies {
    target_fileclass = movie_types;
    action_params { pool = "pool1"; count = 2; }
    condition { last_mod > 6h }
  }
  rule migr_hpc_data {
    target_fileclass = big_hpc_files;
    action_params { pool = "pool2"; count = 16; }
    condition { last_mod > 6h }
  }
  ...
}
```
Implementing “Legacy” Policies

- Modules and templates for “legacy” policies are shipped with robinhood.
- You just need to “include” the right template:

  ```
  %include "templates/lhsm.conf"
  ```

- Then specify policy rules as usual:

  ```
  lhsm_archive_rules {
  ignore_fileclass = noarchive;
  rule archive_daily {
    target_fileclass = myclass1;
    condition { last_archive > 1d or last_mod > 1d}
  }
  ...
  }
  ```
Enhanced Workflows (v3.1)

- **Improved Lustre/HSM workflow**
  - Action rate smoothing/leveling
  - Avoid huge bursts of actions per pass
  - Rate-limited actions

- **Asynchronous accounting**
  - “Accounting” feature: allows retrieving aggregated filesystem stats instantly
    - e.g. stats per user, per HSM status, file size profile, ...
  - Currently, it causes significant performance drop
  - The goal is to make its asynchronous, and possibly distributed
  - Once done, we can implement and provide more aggregated stats, with a limited performance impact:
    - Fine-grained activity tracking per user, per job, ...
Other Performance Enhancements

Expected Lustre improvements

- **Bulk MDT scans**
  - Changelog-like list of entries
  - Expect to dramatically improve initial scan speed

- **New LustreAPI**
  - Work by Cray tracked by LU-5969
  - Optimize massive entry handling
    - Avoid continuous open/close of FS root and “fid” directory for IOCTLs
Conclusion

A new area for Robinhood

- Version 3 is a generic policy framework
  - Schedule what you want!
  - Specific needs can be implemented as additional plugins
  - Allows interactions with site environment

- Major changes are still on the way to:
  - Offer even more features and relevant statistics
  - Improve performance and scalability
  - Support new storage systems

- V3.0 Beta very soon!
  - Scheduled for Q4 2015

- Final V3.0
  - Expected for Q1 2016
Thank you for your attention!

Questions?