Robinhood status and latest improvements

LAD'18

Thomas Leibovici <thomas.leibovici@cea.fr>

September 25th, 2018
Robinhood Big Picture

Input tools

- Posix scan
- Changelog reader
- WIP: Lester connector/JSON import

End-usages

- Policy application
- find/du -like commands
- Reporting CLI
- REST interface & web GUI

Robinhood DB

- MD replicate + addl info
- Aggregated info
**Principle**

- Multi-threaded scan

**Performance**

- ~800k entries/sec on a local FS
- 10 to 80 times slower with Lustre

---

**Robinhood scan speed**

- Speed (entries/sec)
- Scan threads

- **tmpfs**: 790k, 700k
- **xfs**: 72k
- **Lustre 2.5 (POSIX attrs)**: 20k
- **Lustre 2.7 (POSIX attrs)**: 9.3k
- **Lustre 2.5 (+spec attrs)**
- **Lustre 2.7 (+spec attrs)**
PARALLEL SCAN (MULTI-CLIENT)

**Principle**
- Scan can be distributed across multiple clients
- Split the namespace between multiple scanners
- Allows cumulating ops/sec of individual clients

**Performance**

Distributed scan speed

![Graph showing distributed scan speed with different configurations and file systems.](graph.png)

**Example configuration:**
```
scan_only = /dir1;
scan_only = /dir2;
... 
```
Changelog reader

**Principle**

- Read MD updates from Lustre changelogs (CREATE, MKDIR, CLOSE, HSM...) apply to Robinhood DB near real-time
- Changelogs are stored until Robinhood acknowledges them (no event is lost)

**Details**

- 1 reader per MDT
- Possible setup:
  - all readers in a single process  
  - multiple processes on the same host  
  - distributed on multiple hosts

```sh
robinhood --readlog
robinhood --readlog=<mdt_index>
```
### Changelog aggregation

#### Optimization: changelog aggregation

**Without optimization:**

<table>
<thead>
<tr>
<th>Incoming records</th>
<th>Robinhood processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE FID1</td>
<td>CREATE : get info about new entry FID1</td>
</tr>
<tr>
<td>MTIME FID1</td>
<td>MTIME : update info after change (stat)</td>
</tr>
<tr>
<td>TRUNCATE FID1</td>
<td>TRUNCATE : update info after change (stat)</td>
</tr>
<tr>
<td>CLOSE FID1</td>
<td>CLOSE : update info after change (stat)</td>
</tr>
</tbody>
</table>

**Changelog aggregation (since robinhood 2.5): batch processing of changelogs with similar processing**

<table>
<thead>
<tr>
<th>Incoming records</th>
<th>Changelog reader</th>
<th>Robinhood processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE FID1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTIME FID1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUNCATE FID1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSE FID1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Incoming records

Changelog reader
dedup queue

get info about new entry FID1

Robinhood processing
Enhanced Changelog aggregation

With previous optimization:

<table>
<thead>
<tr>
<th>Incoming records</th>
<th>Robinhood processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE FID1</td>
<td>CREATE: get info about new entry FID1</td>
</tr>
<tr>
<td>TRUNCATE FID1</td>
<td>UNLINK: drop the entry from the DB</td>
</tr>
<tr>
<td>CLOSE FID1</td>
<td></td>
</tr>
<tr>
<td>UNLINK FID1</td>
<td></td>
</tr>
</tbody>
</table>

Enhanced changelog aggregation (since robinhood 3.1.4):

<table>
<thead>
<tr>
<th>Incoming records</th>
<th>Changelog reader dedup queue</th>
<th>Robinhood processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE FID1</td>
<td>CREATE FID1</td>
<td></td>
</tr>
<tr>
<td>TRUNCATE FID1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSE FID1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNLINK FID1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Changelog performance

Resulting performance

Example on a scratch filesystem:

- records read = 8889464866
- interesting records = 1533615524
- suppressed records = 7159476712

-80% records

Performance observed: up to 200k records/sec
Performance tips

- Get the latest version of robinhood (3.1.4)
- Increase changelog aggregation window depending on host memory:
  - Parameters “queue_max_size” and “queue_max_age”
- Run changelog reader in a separate process (dedicated memory allocator)
- Run mariadb with jemalloc:
  - In `/etc/systemd/system/mariadb.service.d/override.conf`:

```
[Service]
Environment="LD_PRELOAD=/usr/lib64/libjemalloc.so.1"
```
Work in progress: lester connector for robinhood

- Lester: tool developed by ORNL
  - Directly scans the MDT inode table

- Lester contributions:
  - JSON output
  - dump extra information: link EA, hsm EA

- New robinhood command: rbh-jreader
  - Read JSON output from lester and inject it to robinhood DB
  - File or pipe
Benefits and performances

- Low level ldiskfs scan: fast!
- Takes advantage of distributed metadata

![Diagram showing distribution of inodes across MDTs]

<table>
<thead>
<tr>
<th>Distribution of inodes across MDTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>126 540</td>
</tr>
</tbody>
</table>

Scan duration

<table>
<thead>
<tr>
<th>Lester + rbh-jreader</th>
<th>Namespace scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 h 22 min</td>
<td>33 h 8 min</td>
</tr>
</tbody>
</table>
Policy run enhancements

Enhancements (robinhood 3.1):

- 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...
  - Stackable
  - Plugins (you can implement your own)
  - Provided implementation: “common.rate_limit”
    - Allow limiting the rate of actions (count and/or size)
    - Especially useful for Lustre/HSM archive
Policy schedulers: example

lhsm_archive_parameters {

    # limit archive rate to avoid flooding the MDT coordinator
    schedulers = common.rate_limit;

    rate_limit {
        # max count per period
        max_count = 10000;
        # max size per period: 10GB/s
        max_size = 100GB;
        # period, in milliseconds: 10s
        period_ms = 10000;
    }

...
REST interface

REST API

- Allow querying robinhood DB using a simple HTTP client
- Fine-grained access control (per user, per group, IP, hostname...)
- Package “robinhood-webgui”

Request format:
<server>/api/native/{acct|data|graph}/field.operator/field.operator/...
e.g. sum all information per group:
https://rbh-host/api/native/acct/gid.group/size.desc

```json
[
  {
    "gid": "somegroup",
    "size": "14406475930640792",
    "count": "6813043",
    ...
  },
  ...
]
```

- Full API documentation: /var/www/robinhood/README.txt
Web interface

- New web gui in v3
- File distribution per user, group, size, policy status...
- Filtering
- Namespace browsing

- New features in 3.1.4:
  - Tasks
  - Custom graphs
Web interface: tasks

Scheduled requests (tasks)

- Run custom REST requests at scheduled interval
- Possibility to keep an history of request results, to graph stats evolution
- Target database selection
- 1 click to turn it to a custom graph

Tasks

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Trigger</th>
<th>Request</th>
<th>Action</th>
<th>Database</th>
<th>Last Run</th>
<th>Last execution time</th>
<th>Modify</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>size_by_unit</td>
<td>Hourly</td>
<td>native/entries/whitelist/id.</td>
<td>History</td>
<td>rbh_new</td>
<td>2018-09-18 14:15:12</td>
<td>5 Seconds</td>
<td>Modify</td>
</tr>
<tr>
<td>7</td>
<td>size_by_hunit</td>
<td>Hourly</td>
<td>native/entries/whitelist/id.</td>
<td>History</td>
<td>rbh_new</td>
<td>2018-09-18 14:15:22</td>
<td>5 Seconds</td>
<td>Modify</td>
</tr>
<tr>
<td>11</td>
<td>group_evol2</td>
<td>Hourly</td>
<td>native/acct/gid/group</td>
<td>History</td>
<td>rbh_new</td>
<td>2018-09-18 14:15:17</td>
<td>5 Seconds</td>
<td>Modify</td>
</tr>
<tr>
<td>12</td>
<td>file_by_day</td>
<td>Hourly</td>
<td>native/entries/whitelist/id.</td>
<td>History</td>
<td>rbh_new</td>
<td>2018-09-18 14:15:12</td>
<td>6 Seconds</td>
<td>Modify</td>
</tr>
<tr>
<td>16</td>
<td>file_by_month</td>
<td>Hourly</td>
<td>native/entries/whitelist/id.</td>
<td>Copy</td>
<td>rbh_new</td>
<td>2018-09-18 14:15:01</td>
<td>5 Seconds</td>
<td>Modify</td>
</tr>
</tbody>
</table>

# Name | Trigger | Request | Action | Database | # | # | Add

RUG 2016 | 19 SEPTEMBER 2016
Web interface: custom graphs

Building custom charts

1) In console panel, test your request and see what your graph looks like
2) Give it a name and click “Add custom graph”
Web interface: custom graphs

Custom graphs appear as new items in the menu
Thank you for your attention!

Questions?
Defining a policy

1) Use a template policy, or define a custom policy
   - E.g. lhsm_archive, cleanup, rmdir...

2) Define file classes (based on entry attributes)
   - E.g. name == *.log and (size > 1GB or owner == foo)

3) Define policy rules indicating:
   - Target file classes
   - “last minute” criteria (e.g. time-based)
   - Action and parameters
   Define policy triggers if you plan to run the policy in robinhood daemon

4) Run the policy
   - On all matching entries in the filesystem
   - Or, limited to a user, group, file class, OST set...
Example: pool migration

Example: migrating files to specific pools

1) Policy definition (custom)

```plaintext
define_policy move2pool {
    status_manager = basic;
    scope { type == file and status != ok }
    default_action = cmd("migrate2pool.sh {tgt_pool} {path}");
}
```

2) Define target fileclasses

```plaintext
fileclass wrong_pool_big {
    definition { size > 100GB and ost_pool != "big_pool" }
    move2pool_parameters { tgt_pool = big_pool; }
}
}
fileclass wrong_pool_small {
    definition { size < 10MB and ost_pool != "small_pool" }
    move2pool_parameters { tgt_pool = small_pool; }
}
}
```
Example: pool migration (cont'd)

3) Policy rules:

```plaintext
move2pool_rules {
    rule rulexxx {
        target_fileclass = wrong_pool_small;
        target_fileclass = wrong_pool_big;
        condition {last_mod > 6h}
    }
}
```

4) Running the policy: command examples

```plaintext
# run regularly on all matching entries (daemon)
robinhood --run=move2pool -d

# run once on user foo
robinhood --run=move2pool --target=user:foo

# run once on OST 42
robinhood --run=move2pool --target=ost:42
```
Reporting commands

- **rbh-report**: query robinhood DB (entry info, reports, ...)
- **rbh-find**: “find”-like command to search for entries according to various criteria (attributes, striping, policy status, ...)
- **rbh-du**: “du”-like command with additional filtering features
- **rbh-diff**: display differences between current FS state and rbh DB.