FROM RESEARCH TO INDUSTRY

# Taking back control of HPC file systems with RobinHood Policy Engine

http://robinhood.sf.net

Thomas Leibovici

thomas.leibovici@cea.fr

CEA, DAM, DIF, F-91297 Arpajon, France

LUSTRE ECOSYSTEM WORKSHOP – 2015, MARCH 3-4

www.cea.fr

# LOSING CONTROL? (1)

## Monitoring filesystem contents: what we have

- Filesystem features are limited:
  - df: overall usage
  - Quotas : per user inode/volume usage
- For customized statistics, scanning is often required:
  - find, du…

➔ **Become endless as filesystems grow**

# LOSING CONTROL? (2)

## Monitoring filesystem contents: what we want

- An accurate view of user data profiles:
  - Size profile
  - Age profile
  - Per project, per group, per directory…
- Aggregated statistics at will
  - Available in a few seconds
- Customized reports
  - Multiple arbitrary criteria
- Filesystem activity indicators

# LOSING CONTROL? (3)

## Data management: what we have

- Tools:
  - cp, rsync, backup tools, …
- Policies:
  - find <criteria> -exec <action>

➔ **Again, needs scanning**

➔ **Managing multiple criteria/actions is even longer and painful!**

# LOSING CONTROL? (4)

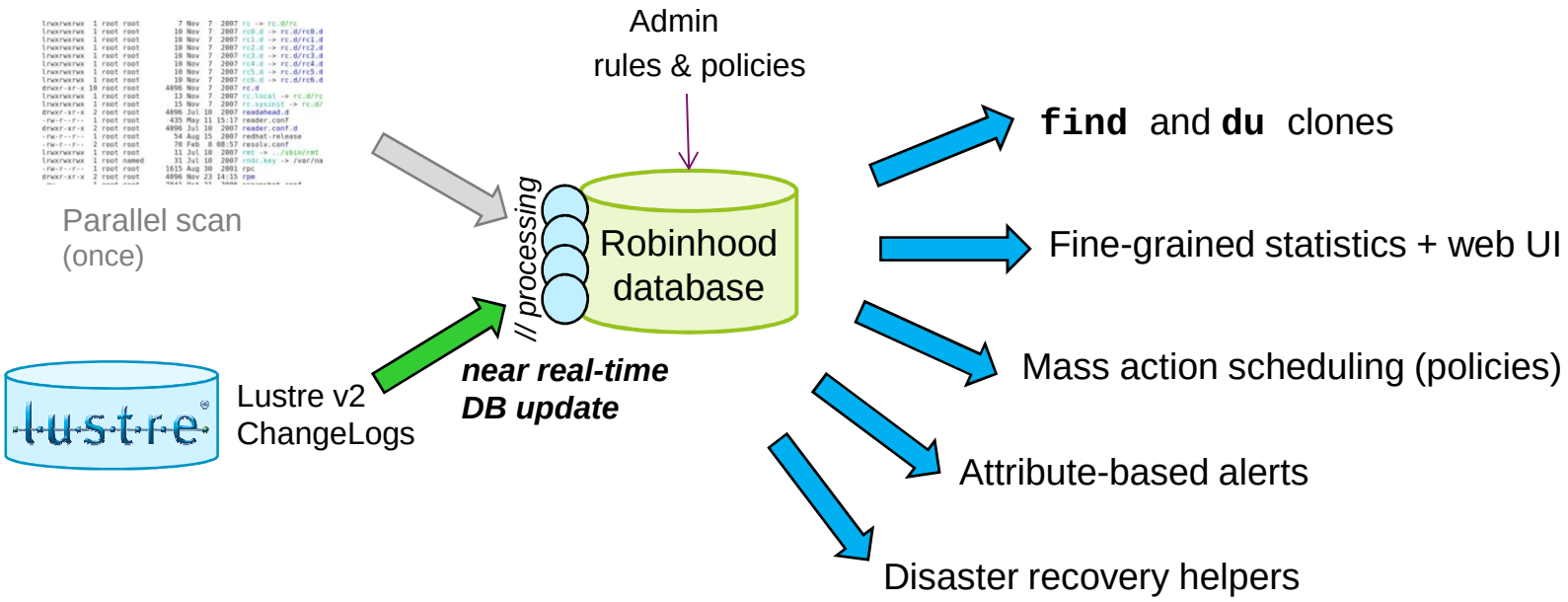
## Data management: what we want

- Applying mass actions to filesystem entries
  - Fast
  - Using various criteria and actions
- Life cycle management
  - HSM migration
  - Pool to pool migration
  - Cleaning old unused data
  - ...

# HOW ROBINHOOD CAN HELP

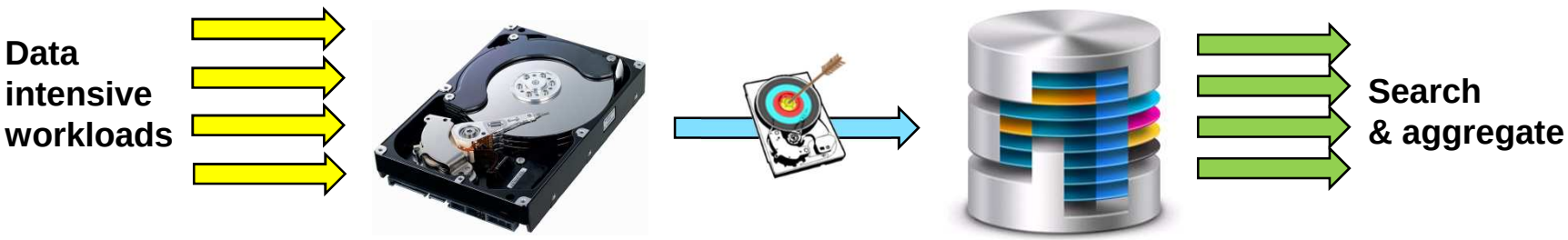
# ROBINHOOD POLICY ENGINE

## Big picture



# FILESYSTEM AND DATABASE BENEFITS

## Benefits




### Filesystem

Goals:

- Optimize data access
  - Bandwidth, data allocation
- Optimize medatada access for POSIX:
  - lookup/readdir/create/unlink

```
lfs find . -user foo –size -1024 | wc -l
```

### Database

Goals:

- Optimize per-record access
  - select/insert/update
- Optimize multi-criteria searches
- Optimize aggregating/sorting information

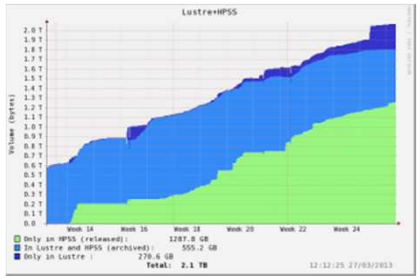
```
select count(*) from ENTRIES where
user='foo' and size<1024
```

# RBH-REPORT

## Examples of reports

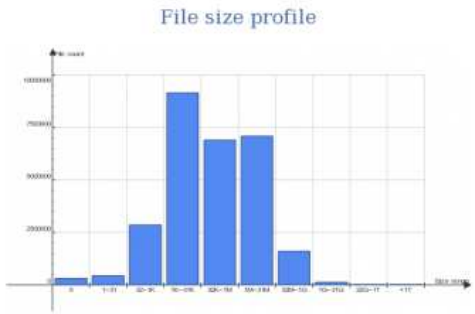
- Inode count and volume usage:
  - Per user, per group, per type, per HSM status, both…

```
$ rbh-report -u foo* -S
user , group,   type,  count, spc_used, avg_size
foo1 , proj001, file, 422367, 71.01 GB, 335.54 KB
…
Total: 498230 entries, 77918785024 bytes used (72.57 GB)
```




- File size profiles per user, per group…

```
$ rbh-report --szprof -i|-u 'foo*'|-g 'bar*'
```




- Top users, top groups, top file sizes, top directories…

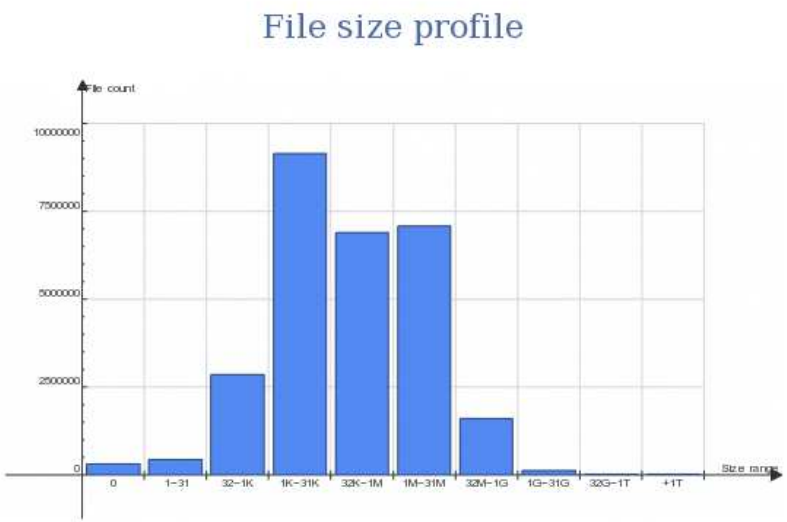
```
$ rbh-report --topdirs
```

- Changelog statistics (per-operation) : CREATE/sec, UNLINK/sec, …
- Oldest files

# WEB INTERFACE

## Web UI

***Usage stats (per user, per group)***




***File size profile (global / per user / per group)***

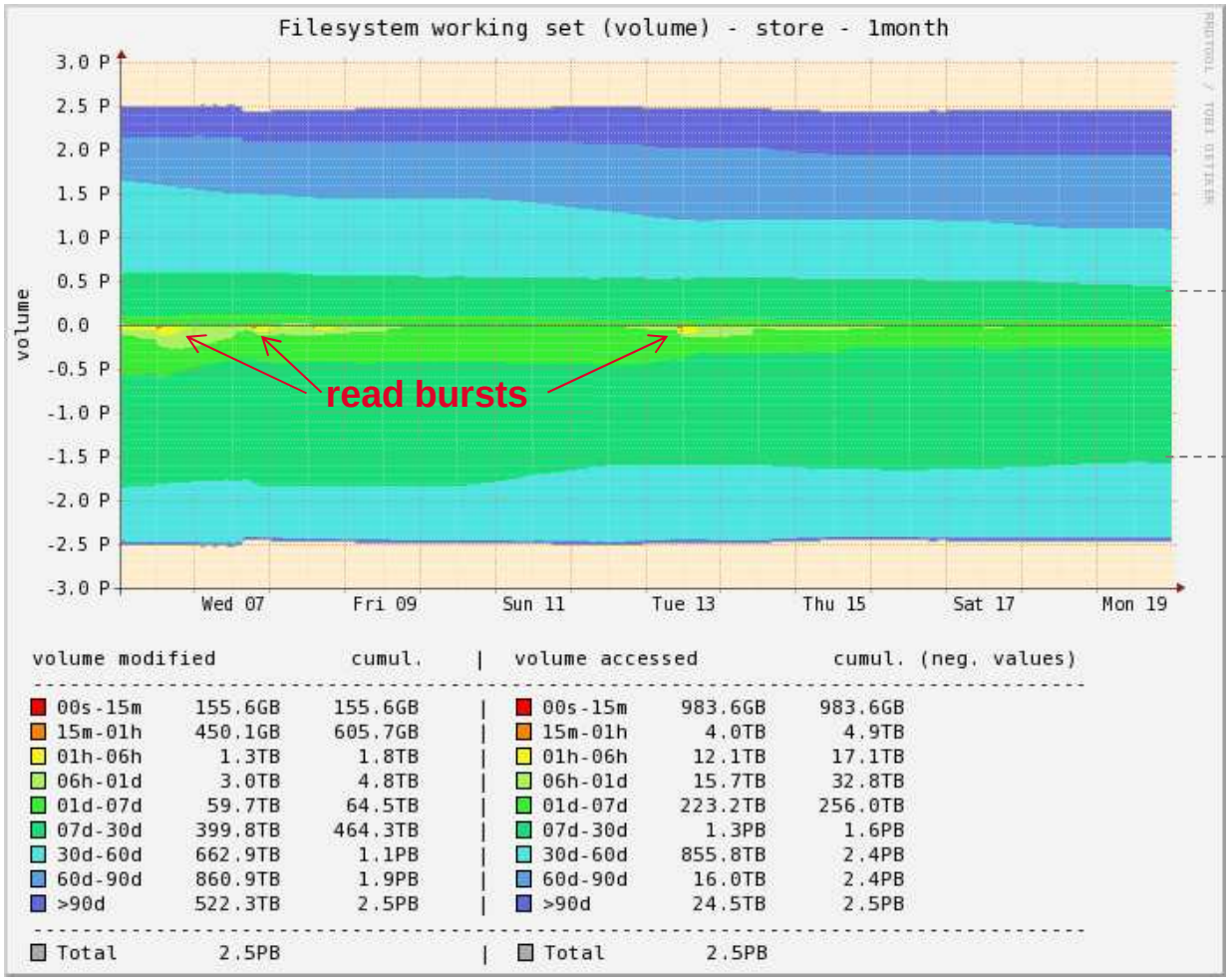
| User | Total volume | File count | Avg file size | file size ratio | | | |
|---|---|---|---|---|---|---|---|
| | | | | empty | <1K | <1M | <1G |
| | 11.98 TB | 36 251 | 346.48 MB | 0.74% | 3.17% | 67.12% | 97.65% |
| | 15.02 TB | 86 584 | 181.94 MB | 21.73% | 36.00% | 86.33% | 93.63% |
| | 120.02 TB | 114 445 | 1.07 GB | 13.30% | 30.75% | 65.83% | 75.80% |
| | 27.31 TB | 65 944 | 434.31 MB | 21.16% | 30.35% | 75.72% | 84.30% |
| | 7.53 TB | 27 725 | 284.86 MB | 0.76% | 26.44% | 80.27% | 90.80% |
| | 19.17 TB | 87 617 | 229.41 MB | 9.19% | 22.32% | 61.84% | 81.17% |
| | 74.17 TB | 38 685 | 1.96 GB | 16.62% | 22.65% | 50.07% | 60.83% |
| | 57.35 TB | 112 730 | 533.41 MB | 1.97% | 6.21% | 36.07% | 65.95% |

# CUSTOM QUERIES

## Filesystem “temperature”

data **production**
(mod. time)

data **usage**
(last access)




1 month
working set

# RBH-FIND, RBH-DU

## Fast *find* and *du* clones

- Query robinhood DB instead of performing POSIX namespace scan

➔ faster!

```
$ rbh-find –user ”foo*” –size +1G –ost 4
```

  - 20 sec for 40M entries (vs. hours for ‘lfs find’)

- Enhanced *du* :
  - Detailed stats (by type…)
  - Can filter by user

```
$ rbh-du -sH /fs/dir –u foo --details
/fs/dir
          symlink count:30777,   size:1.0M, spc_used:9.1M
          dir     count:598024,  size:2.4G, spc_used:2.4G
          file    count:3093601, size:3.2T, spc_used:2.9T
```

# LUSTRE-SPECIFIC FEATURES

## Lustre specific features

- Changelogs
  - Near real-time DB update.
  - Avoids FS scans.
- Access entries by FID
  - Reduces POSIX overhead
  - Insensitive to rename's.
- OSTs aware
  - Monitoring individual OST usage and triggers purges per OST.
- Striping and pools
  - Allow querying entries per stripe info.
  - List impacted entries in case of OST disaster.
- HSM support
  - Aware of Lustre/HSM flags and HSM changelog records.
  - Triggers Lustre/HSM actions

## Robinhood supports all lustre versions since 1.8

# HELPING WHEN A DISASTER OCCURS

- Robinhood can indicate impacted entries when an OST is lost/corrupted:
  - Lists entries striped on a given OST
  - Indicates if entries had data in these stripes

```
> rbh-report --dump-ost 2,5-8
type,       size,                    path, stripe_cnt, stripe_size,                      stripes, data_on_ost[2,5-8]
file,    8.00 MB,        /fs/dir.1/file.8,          2,     1.00 MB,  ost#2: 797094, ost#0: 796997,          yes
file,   29.00 MB,        /fs/dir.1/file.29,         2,     1.00 MB,  ost#2: 797104, ost#0: 797007,          yes
file,    1.00 MB,        /fs/dir.4/file.1,          2,     1.00 MB,  ost#3: 797154, ost#2: 797090,           no
file,   27.00 MB,        /fs/dir.1/file.27,         2,     1.00 MB,  ost#3: 797167, ost#2: 797103,          yes
file,   14.00 MB,        /fs/dir.5/file.14,         2,     1.00 MB,  ost#3: 797161, ost#2: 797097,          yes
file,   13.00 MB,        /fs/dir.7/file.13,         2,     1.00 MB,  ost#2: 797096, ost#0: 796999,          yes
file,   24.00 KB,        /fs/dir.1/file.24,         2,     1.00 MB,  ost#1: 797102, ost#2: 797005,           no
…
```

File is not impacted:
it must just be restriped to
sane OSTs.

File data is impacted:
admin must delete it and notify the user.
HSM: restore the file from the archive.

# POLICIES (TODAY)

## Robinhood v2.5 flavors and policies

| Mode | POSIX | Lustre | "migration" policy | "purge“ policy | "hsm_remove" policy | "rmdir" policy |
|---|---|---|---|---|---|---|
| **robinhood-tmpfs** | ✓ | ✓ | - | rm (old files) | - | rmdir, rm –rf |
| **robinhood-backup** | | ✓ | Copy to storage backend | - | rm in storage backend | - |
| **robinhood-lhsm** | | ✓ | Lustre HSM archive | Lustre HSM release | Lustre HSM remove | - |

## Policy example

```
fileclass BigLogFiles {
    definition { type == file and size > 100MB
                 and (path == /fs/logdir/*
                      or name == *.log) }
 ...
}
```

```
purge_policies {
    ignore_fileclass = my_files;

    policy purge_logs {
        target_fileclass = BigLogFiles;
        condition { last_mod > 15d }
    }
}
```

# ADDRESSING TODAY AND FUTURE CHALLENGES

# CHALLENGES

Robinhood challenges include:

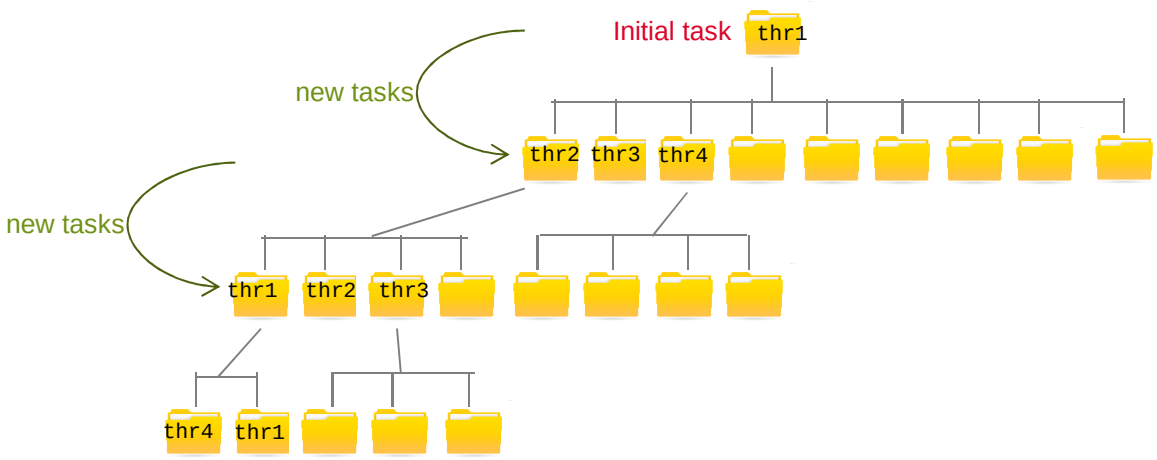
- Collecting
- Processing
- Storing
- Aggregating
- Reporting
- Managing data
- Adapting to new storage architectures

*more and more entries*

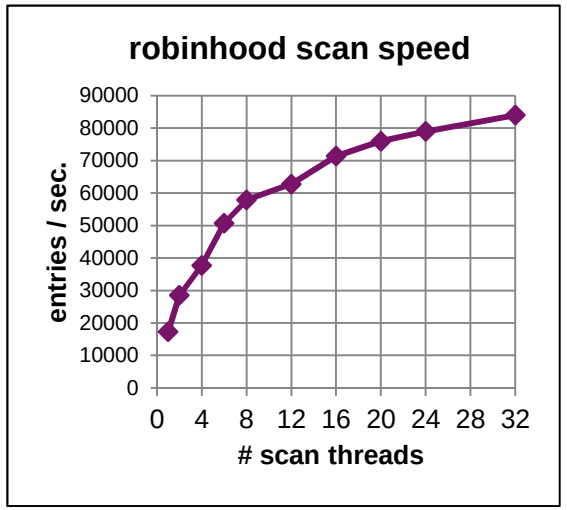
# COLLECTING: CHALLENGE #1

## Scanning

- Even with Lustre changelogs, an initial scan is still needed
  - The faster the better!
- Robinhood implements a multi-threaded scan algorithm
  - The namespace is split into individual tasks
  - Each task consists in scanning a single directory
  - Depth-first traversal to limit memory usage




*Example with 4 threads*

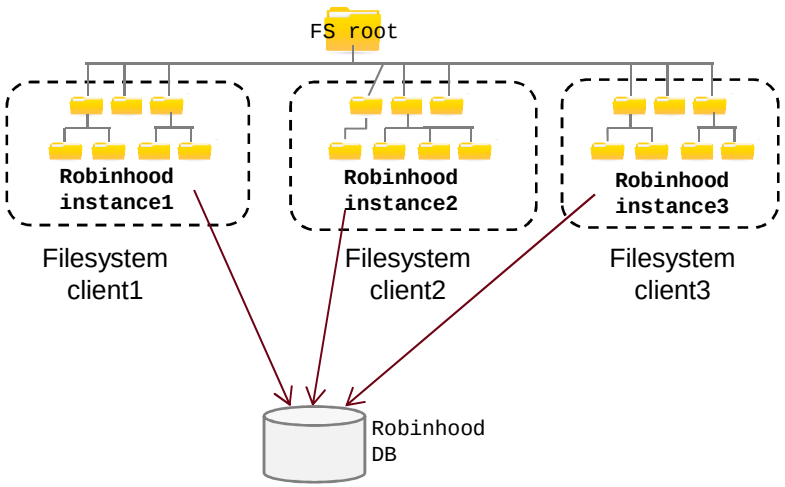



*XFS filesystem scan*

# SCANNING (CONT'D)

## Distributed scanning

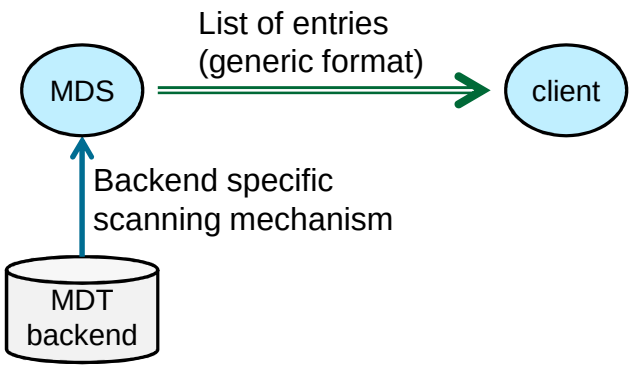
- Robinhood allows distributing the scan across multiple clients
  - Admin must split the namespace between instances
- This allows cumulating ops/sec of individual clients
- All instances feed the same DB



# SCANNING: PERSPECTIVES

## Future: bulk scan as a filesystem service?

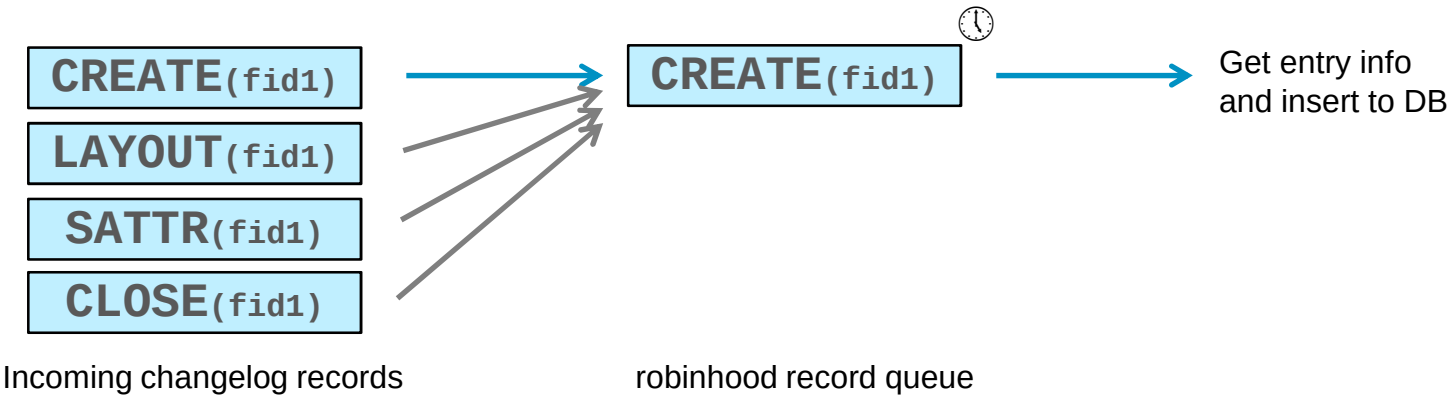
- Robinhood scan is based on POSIX calls
- ext4 low level scan is faster than a POSIX traversal
- But: basing robinhood scan on e2scan would make it very dependant of Lustre MDT backend.
- Proposal: implement a “bulk scan” feature, similar to changelog streams
  - The client opens a steam to receive the list of all entries in the filesystem in an arbitrary order.



# COLLECTING: CHALLENGE #2

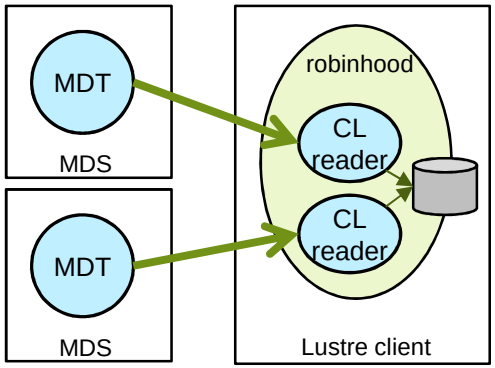
## Changelogs: reducing workload by 85%!

- Information from Changelog records may be redundant:
  - Writing a file triggers several events: CREATE, LAYOUT, MTIME, SATTR, CLOSE...
  - But robinhood only needs to update entry information once
- Implemented solution: changelog batching mechanism
  - Robinhood delays changelog processing to batch redundant records in memory
    - Configurable delay, configurable max delayed records
  - Multiple redundant records for the same entry are batched
  - Dramatically decrease the incoming changelog throughput and operation rate on robinhood DB ➔ **-85%** observed in real life

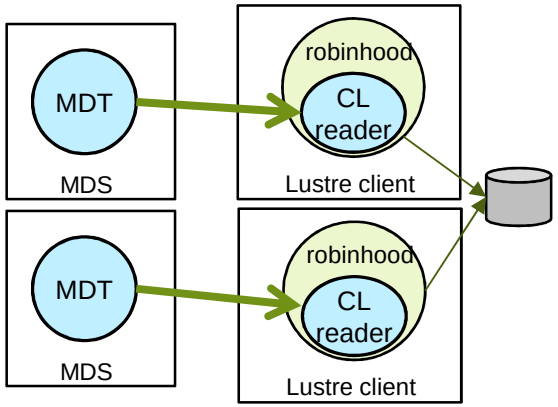


# DNE: processing multiple changelog streams

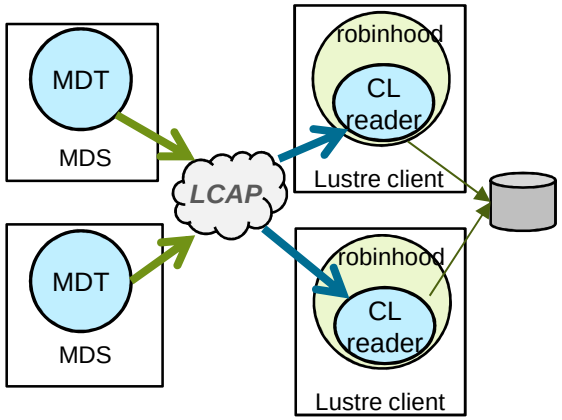
- Possible configurations (today):
  - 1 reader thread per MDT
  - 1 reader process per MDT (possibly on multiple clients)
  - N readers with LCAP Changelog proxies




*1 robinhood instance, 1 thread / MDT*




*1 robinhood instance per MDT*




*Using LCAP proxies*

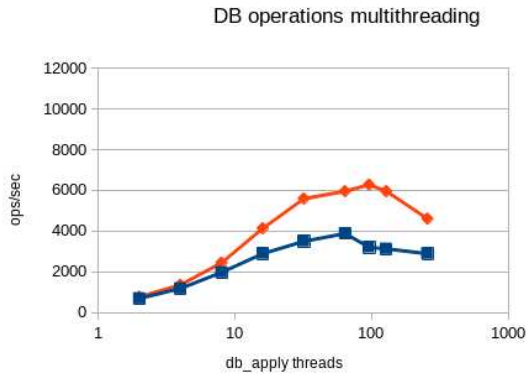
➔ Database: need for horizontal scalability
- Evaluating distributed databases (sharding, …)

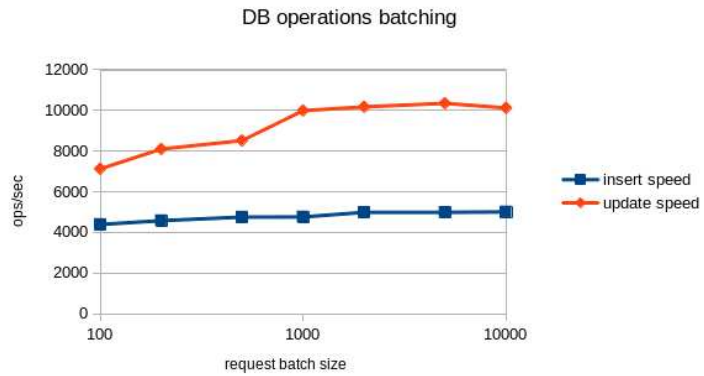
# Optimizing database ingest rate

- 2 implemented strategies:
  - Perform multiple DB operations in parallel
  - Batch database operations into large transactions to reduce DB IOPS

"Slow" DB backend (spinning disk):

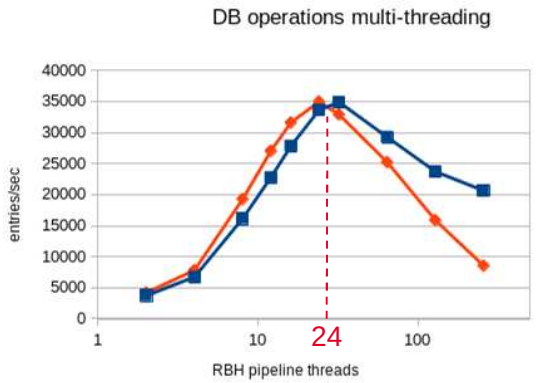
➔ Best result: batching (~10.000 entries/sec)

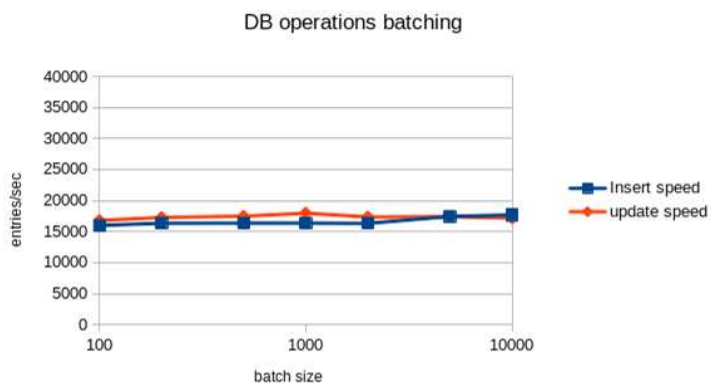



"Fast" DB backend (SSD):

➔ Best result: multi-thread (~35.000 entries/sec)

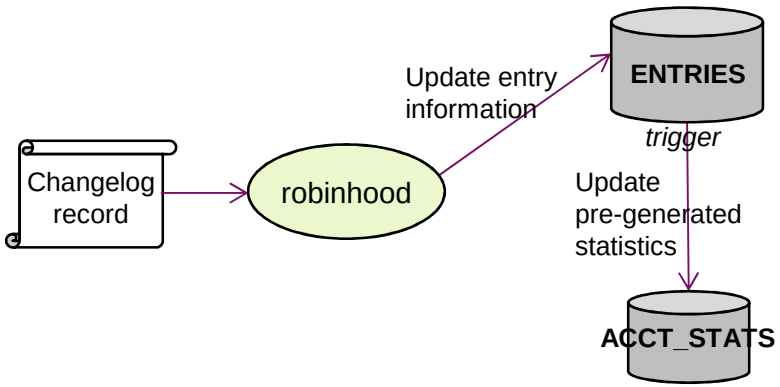


# AGGREGATING & REPORTING

## Maintaining aggregated statistics

- "select user, sum(size) from ENTRIES group by user"
  - take minutes for a billion records
  - ➔ unacceptable for a Web UI!
- To allow retrieving statistics instantly, robinhood maintains some statistics on-the-fly
  - Inode, volume, size profiles per user, per group, per status, …
  - Update is done in the same DB transaction to avoid inconsistent stats
    - But : impacts insert/update performance (wide locking of stat table)
    - More stat tables ➔ more impact

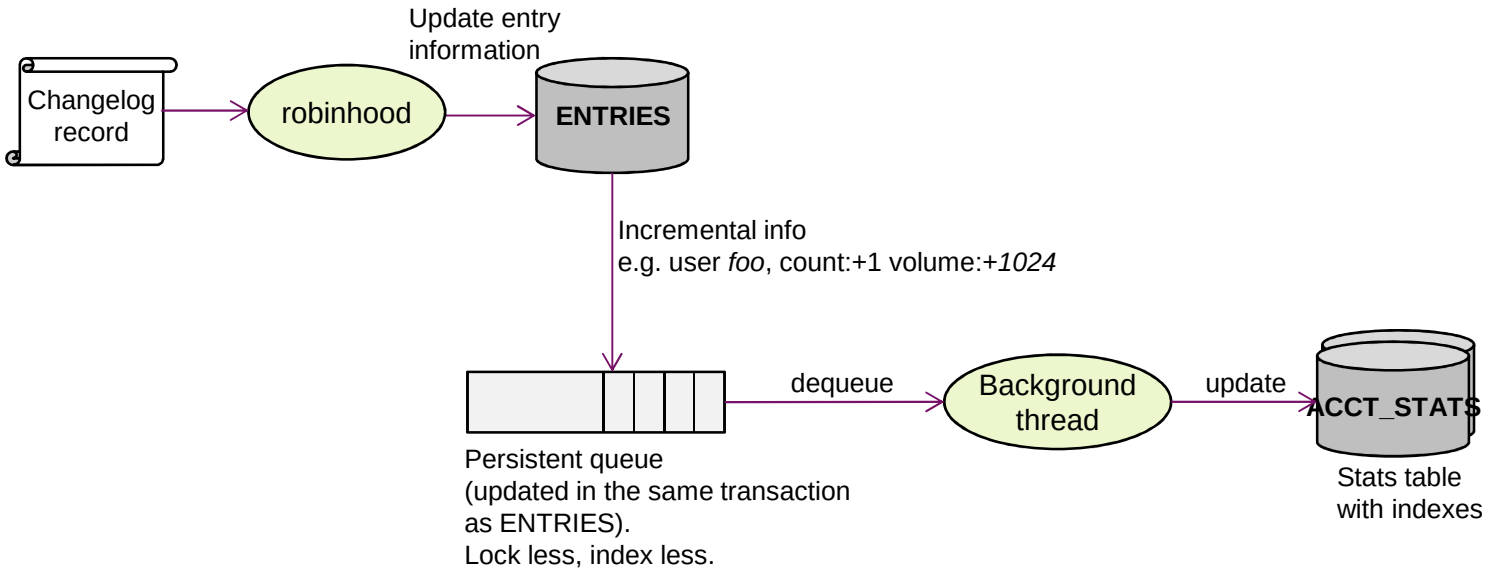



| fid, | owner, | group, | type, | size, | last_access, … |
|---|---|---|---|---|---|
| *xx1,* | *user1,* | *grp1,* | *file,* | *3192,* | *1424703285* |
| *xx2,* | *user1,* | *grp1,* | *file,* | *239840,* | *1325324324* |
| *xx3,* | *user1,* | *grp1,* | *file,* | *0,* | *1339324907* |
| *xx4,* | *user2,* | *grp24,* | *dir,* | *4096,* | *1334343443* |
| *xx5,* | *user2,* | *grp24,* | *dir,* | *4096,* | *1423434276* |
| … | | | | | |

| owner, | group, | type, | count, | volume, … |
|---|---|---|---|---|
| *user1,* | *grp1,* | *file,* | *272289,* | *5372837784* |
| *user1,* | *grp1,* | *dir,* | *4324,* | *20437* |
| *user2,* | *grp3,* | *file,* | *24,* | *12448493* |
| *user2,* | *grp3,* | *symlink,* | *234,* | *7891* |
| … | | | | |

## Reducing the impact of aggregated statistics

- Asynchronous, near real-time stats update



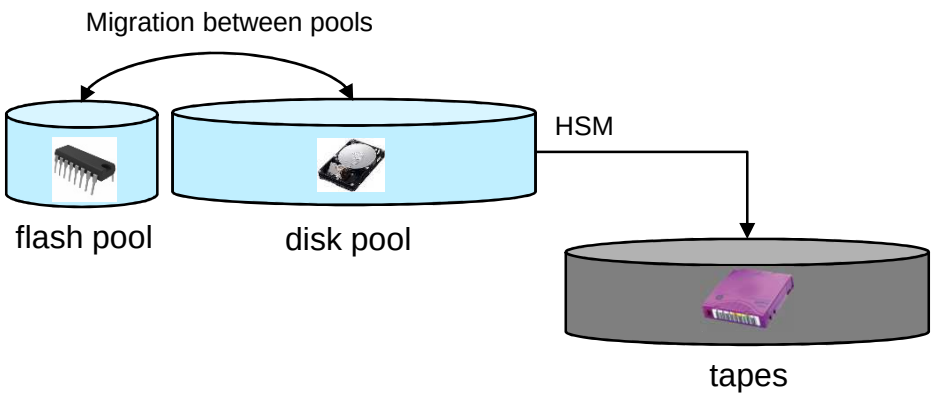

- Stats are updated near-real time
- Stats are eventually consistent (no increment is missed)
- More stat tables doesn't impact main insert/update stream (from changelog)

# ADAPTING TO NEW ARCHITECTURES

## Managing heterogeneous filesystems

- Yesterday: most Lustre filesystems were homogeneous (disks only)
- Tomorrow: most Lustre filesystems will be heterogeneous (disks, flash, …)

- Today's robinhood HSM policies are limited to 2 levels:
  - Archive to backend storage
  - Release from Lustre level
- An evolution is needed to manage more levels:
  - Allow fine-grained scheduling of migrations between several pools + HSM




for this…

# GENERIC POLICIES

## Robinhood v3 generic policies

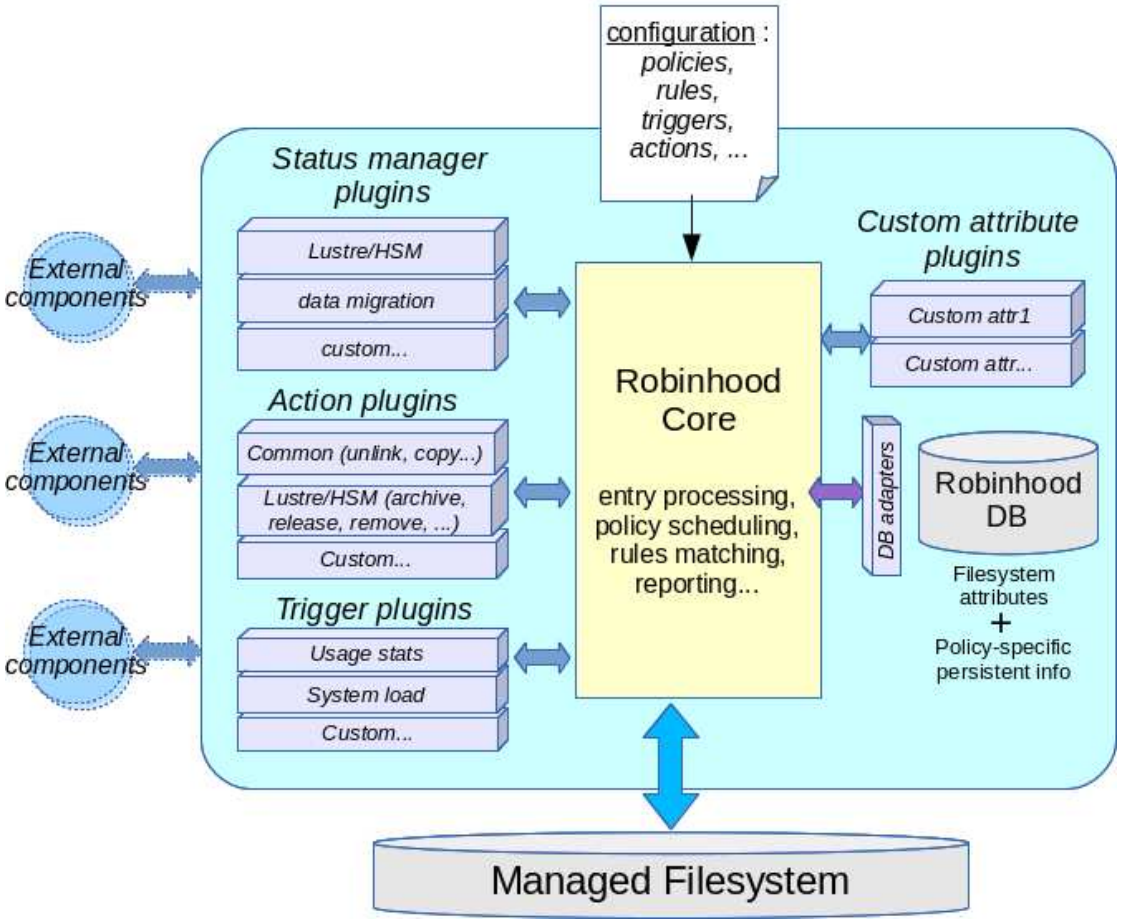
- Robinhood v2.5:
  - Limited set of policies, statically defined
  - 1 mode = 1 package = 1 specific set of commands

| Package | "migration" policy | "purge“ policy | "hsm_remove" policy | "rmdir" policy |
|---|---|---|---|---|
| **robinhood-tmpfs** | - | rm (old files) | - | rmdir, rm –rf |
| **robinhood-backup** | Copy to storage backend | - | rm in storage backend | - |
| **robinhood-lhsm** | Lustre HSM archive | Lustre HSM release | Lustre HSM remove | - |

- Robinhood v3.0: generic policies
  - 1 single generic mode for all purposes
  - Can implement all “legacy” policies (config-defined)
  - Can implement new policies at will (config-defined)

| Package | Generic policies |
|---|---|
| **robinhood** | Fully configurable |

# ROBINHOOD V3: BIG PICTURE




- Robinhood core made generic:
  - Purpose-specific code moved out of robinhood core: now dynamic plugins loaded at run-time
  - All policy behaviors made configurable
  - Users can write their own plugins for specific needs
- Easily implement new policies, just by writing a few lines of configuration
  - OST rebalancing
  - Pool-to-pool data migration
  - Data integrity checks
  - Trash can mechanism
  - Massive data conversions
  - …

# ROBINHOOD V3: ROADMAP

## Planned features (v3.0 and v3.x)

- Generic policies (generic core, plugin based-architecture…)
- New fileclass management
- Asynchronous aggregated stats
- New aggregated stats:
  - changelog counters per user, per job…
  - Instant fileclass summary
  - Instant ‘du’ for a given level of the namespace
- New policy trigger types
- Customizable attributes
- Support new DB engines (PGSQL…)

## Release plans

- V3.0-beta1: 3Q2015
- V3.0: 4Q2015

# BEYOND ROBINHOOD V3

## Future plans

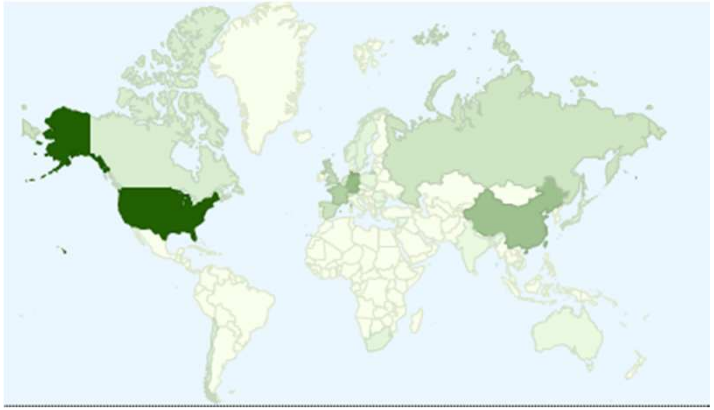
- Distributed database
- Full asynchronous processing model
- Support new types of storage systems
  - e.g. object stores (non POSIX)

# ABOUT THE PROJECT

## A few words about the project

- Project started in 2006
- OpenSource since 2009 (LGPL compatible)
- User community 100-200 sites (~80% on Lustre)

- Git repository:
  - http://github.com/cea-hpc/robinhood
- Project home page:
  - http://robinhood.sf.net
- Mailing lists:
  - robinhood-support@lists.sf.net
  - robinhood-devel@lists.sf.net
  - robinhood-news@lists.sf.net



# WRAP UP

## Summary

- Robinhood is a swiss-army knife to manage filesystems
  - to monitor filesystem contents
  - to schedule automatic actions on filesystem entries
- It continuously evolves:
  - to support and take advantage of new Lustre features
  - to maintain/provide new useful stats to sysadmins
  - to be prepared to new generations of storage systems

## Conclusion

- Drop your old-fashioned scripts based on ‘find’ and ‘du’

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