Robinhood PolicyEngine

First steps (tutorial)

Backup mode for Lustre 2.x

v2.5.2

May 21st, 2014

<robinhood-support@lists.sourceforge.net>

Table of contents

The purpose of this document is to guide you for the first steps of robinhood-backup installation and configuration. For now, this mode is only supported for Lustre 2.x filesystems. We will first run a very simple instance of Robinhood for statistics and monitoring purpose only. Then we will configure it for archiving files. Finally we will deal with more advanced usage, by defining filesets and associating different migration policies to them, undeleting files…

1. Installation

1.1. Robinhood

1.1.1. Install from RPM

Pre-generated RPMs can be downloaded on sourceforge, for the following configurations:
- x86_64 architecture , RedHat 5/6 Linux families
- MySQL database 5.x
- Lustre 2.1, 2.2, 2.3, 2.4, 2.5

Purpose specific RPM: `robinhood-backup`
It must be installed on a Lustre client. It is recommended to run the same Lustre version on this client and Lustre servers.
It includes:
- 'rbh-backup' daemon
- Archive management tools: 'rbh-backup-import', 'rbh-backup-undo-rm', 'rbh-backup-recov', 'rbh-backup-rebind'
- Reporting commands: 'rbh-backup-report', 'rbh-backup-find', 'rbh-backup-du', 'rbh-diff'.
- Copy helpers: 'rbhext_tool' (simple cp wrapper), 'rbhext_tool_clnt' and 'rbhext_tool_svr' (for using multiple copy nodes).
- configuration templates
- /etc/init.d/robinhood-backup init script

Admin RPM (all purposes): `robinhood-adm`
Includes 'rbh-config' configuration helper. It is helpful on the DB host and Lustre MDS.

`robinhood-webgui` RPM installs a web interface to visualize stats from Robinhood database. It must be installed on a HTTP server.
See section 7 for more details about this interface.

1.1.2. Build and install from the source tarball

1.1.2.a. Requirements

Before building Robinhood, make sure the following packages are installed on your system:
- mysql-devel
- lustre API library (if Robinhood is to be run on a Lustre filesystem): '/usr/include/liblustreapi.h' and '/usr/lib*/liblustreapi.a' are installed by lustre rpm.

1.1.2.b. Build

Retrieve Robinhood tarball from sourceforge: http://sourceforge.net/projects/robinhood

Unzip and untar the sources:

```
tar zxvf  robinhood-2.5.2.tar.gz
cd robinhood-2.5.2
```

Then, use the "configure" script to generate Makefiles:

- use the `--with-purpose=BACKUP` option for using it for backup purpose;

```
./configure --with-purpose=BACKUP
```

Other '`./configure`' options:

- You can change the default prefix of installation path (default is /usr) using: '`--prefix=<path>`'

Then build the RPMs:

```
make rpm
```

RPMs are generated in the 'rpms/RPMS/<arch>' directory. RPM is possibly tagged with the lustre version it was built for.

1.2. MySQL database

Robinhood needs a MySQL database for storing its data. This database can run on a different host from Robinhood node. However, a common configuration is to install robinhood on the DB host, to reduce DB request latency.

1.2.1. Requirements

Install *mysql* and *mysql-server* packages on the node where you want to run the database engine.

Start the database engine:

```
service mysqld start
```

1.2.2. Creating database

With the helper script:

To easily create robinhood database, you can use the 'rbh-config' script. Run this script on the database host to check your system configuration and perform database creation steps:

```
# check database requirements:
rbh-config precheck_db

# create the database:
rbh-config create_db
```

Note: if no option is given to `rbh-config`, it prompts for configuration parameters interactively. Else, if you specify parameters on command line, it runs in batch mode.

Write the database password to a file with restricted access (root/600),
e.g. `/etc/robinhood.d/.dbpassword`

or manually:

Alternatively, if you want a better control on the database configuration and access rights, you can perform the following steps of your own:

- Create the database (one per filesystem) using the `mysqladmin` command:
  `mysqladmin create <robinhood_db_name>`

- Connect to the database:
  `mysql <robinhood_db_name>`

  Then execute the following commands in the MySQL session:

  - `GRANT USAGE ON robinhood_db_name.* TO 'robinhood'@'%'`
    `identified by 'password';`
  - `GRANT ALL PRIVILEGES ON robinhood_db_name.* TO 'robinhood'@'%'`
    `identified by 'password';`
  - The 'super' privilege is required for creating DB triggers (needed for accounting optimizations):
    `GRANT SUPER ON *.* TO 'robinhood'@'%' IDENTIFIED BY 'password'`
    `;`
  - Refresh server access settings:
    `FLUSH PRIVILEGES ;`

  - You can check user privileges using:
    `SHOW GRANTS FOR robinhood ;`

- For testing access to database, execute the following command on the machine where robinhood will be running :
  `mysql --user=robinhood --password=password --host=db_host`
  `robinhood_db_name`

  If the command is successful, a SQL shell is started. Else, you will get a 'permission denied' error.

At this time, the database schema is empty. Robinhood will automatically create it the first time it is launched.

2. First run

The best way to use robinhood on Lustre v2 is reading MDT changelogs. This Lustre feature makes it possible to update Robinhood database near real-time. Scanning the filesystem is no longer required after the initial filesystem scan (still needed to populate the DB).

In any case, robinhood runs on a lustre client.

2.1 Activate MDT Changelogs

If filesystem MDS and MGS are on the same host, you can simply enable this feature by running ‘rbh-config’ on this host (for other cases, see *Robinhood admin guide*). 'rbh-config' is installed by robinhood-adm package.

```
rbh-config enable_chglogs
```

This registers a changelog reader as ‘cl1’ and sets the changelog event mask (see `/proc/fs/lustre/mdd/<fsname>-*/changelog_mask`).
The reader is registered persistently. However, the changelog mask must be set when restarting the MDS (prior to any filesystem operation).

2.2 Simple configuration file

Let’s start with a basic configuration file:

```
General {
    fs_path = "/lustre";
}
Log {
    log_file       = "/var/log/robinhood/lustre.log";
    report_file    = "/var/log/robinhood/reports.log";
    alert_file     = "/var/log/robinhood/alerts.log";
}
ListManager {
    MySQL {
        server = db_host;
        db = robinhood_test;
        user = robinhood;
        password_file = /etc/robinhood.d/.dbpassword;
    }
}

Backend {
    root          = "/archive";
    mnt_type      = nfs;
}

ChangeLog {
   MDT {
       mdt_name  = "MDT0000";
       reader_id = "cl1";
   }
}
```

General section:
‘fs_path’ is the mount point of the lustre filesystem we want to manage.

Log section:
Make sure the log directory exists.

Note1: you can also specify special values ‘stderr’, ‘stdout’ or ‘syslog’ for log parameters.
Note2: robinhood is compliant with log rotation (if its log file is renamed, it will automatically switch to a new log file).

ListManager::MySQL section:
This section is for configuring database access.
Set the host name of the database server (*server* parameter), the database name (*db* parameter), the database user (*user* parameter) and specify a file where you wrote the password for connecting to the database (*password_file* parameter).
/!\ Make sure the password file cannot be read by any user, by setting it a ‘600’ mode for example.
If you don’t care about security, you can directly specify the password in the configuration file, by setting the *password* parameter.
E.g.: `password = ‘passw0rd’ ;`

Backend section:
This defines your archive backend. Its namespace must be accessible from the robinhood node as a POSIX filesystem. In this example we use a NFS filesystem mounted as `/archive` (*root* and *mnt_type* parameters).

ChangeLog::MDT section:
This section controls Changelog reading. For this simple case, we only specify the *mdt_name* (always ‘MDT0000’ if you don’t use DNE with multiple MDTs) and the registered Changelog reader as *reader_id* (usually ‘cl1’ if you have a single Changelog consumer).

2.3 Running initial scan

To populate the DB, we need to run an initial scan. Unlike scanning in daemon mode, we just want to scan once and exit. Thus, we run rbh-backup with the ‘--scan’ and ‘--once’ option.
You can specify the configuration file using the ‘–f’ option, else it will use the config file in ‘`/etc/robinhood.d/backup`’. If you have several config files in this directory, you can use a short name to distinguish them. e.g. ‘`-f test`’ for ‘`/etc/robinhood.d/backup/test.conf`’.
If you want to override configuration values for log file, use the ‘–L’ option. For example, let’s use ‘-L stdout’

```
rbh-backup –f test –L stdout --scan ––once
```
or just:
```
rbh-backup –L stdout --scan ––once
```
(if your config file is the only one in /etc/robinhood.d/backup)

You should get something like this:

```
2013/07/17 13:49:06: FS Scan | Starting scan of /mnt/lustre
2013/07/17 13:49:06: FS Scan | Full scan of /mnt/lustre completed, 7130
entries found. Duration = 0.07s
2013/07/17 13:49:06: FS Scan | File list of /mnt/lustre has been updated
2013/07/17 13:49:06: Main | All tasks done! Exiting.
```

2.4 Reading Lustre Changelogs

Then we want to read MDT changelogs to keep the DB up-to-date. We start it as a daemon, as we want to this continuously:

```
rbh-backup --readlog –detach
```

3 Getting filesystems statistics

Now the DB is updated near real-time, we can get fresh statistics about the filesystem.

3.1 rbh-backup-report

Now we performed a scan, we can get stats about users, files, directories, etc. using `rbh-backup-report`:

- Get stats for a user: -u option

`rbh-backup-report –u foo`

```
user      ,     type,      count,   spc_used,   avg_size
foo       ,      dir,      75450,  306.10 MB,    4.15 KB
foo       ,     file,     116396,   11.14 TB,  100.34 MB

Total: 191846 entries, 12248033808384 bytes used (11.14 TB)
```

- Split user's usage per group: -S option

`rbh-backup-report –u bar –S`

```
user      ,      group,     type,      count,   spc_used,   avg_size
bar       ,      proj1,     file,          4,    40.00 MB,   10.00 MB
bar       ,      proj2,     file,       3296,   947.80 MB,  273.30 KB
bar       ,      proj3,     file,     259781,   781.21 GB,    3.08 MB
```

- Get largest files: --top-size option

`rbh- backup-report –-top-size`

```
rank,           path,       size, user, group,          last_access,            last_mod,  migr. class
   1, /tmp/file.big1,  512.00 GB, foo1,   p01,  2012/10/14 17:41:38,  2011/05/25 14:22:41,     BigFiles
   2, /tmp/file2.tar,  380.53 GB, foo2,   p01,  2012/10/14 21:38:07,  2012/02/01 14:30:48,     BigFiles
   3, /tmp/big.1,      379.92 GB, foo1,   p02,  2012/10/14 20:24:20,  2012/05/17 17:40:57,     BigFiles
...
```

- Get top space consumers: --top-users option

`rbh- backup-report –-top-users`

```
rank, user      ,   spc_used,      count,  avg_size
   1, usr0021   ,   11.14 TB,     116396,  100.34 MB
   2, usr3562   ,    5.54 TB,        575,    9.86 GB
   3, usr2189   ,    5.52 TB,       9888,  585.50 MB
   4, usr2672   ,    3.21 TB,     238016,   14.49 MB
   5, usr7267   ,    2.09 TB,       8230,  266.17 MB
```

```
...
```

Notes:

- '--by-count' option sorts users by entry count
- '--by-avgsize' option sorts users by average file size
- '--reverse' option reverses sort order (e.g. smallest first)
- Use '--count-min *N*' option to only display users with at least *N* entries.
- '--by-size-ratio' option makes it possible to sort users using the percentage of files in the given range.

- Filesystem content summary: -i option

`rbh-backup-report –i`

```
status , type    ,      count,     volume,   avg_size
n/a    , dir     ,    1780074,    8.02 GB,    4.72 KB
new    , symlink ,     496142,   24.92 MB,         53
new    , file    ,   21366275,   91.15 TB,    4.47 MB

Total: 23475376 entries, 100399805708329 bytes (91.31 TB)
```

This report indicates the count and volume of each object type, and their status.
As we have not archived data for now, all objects are marked as 'new'.
This field does not make sense for directory objects (n/a), as they do not store data by themselves.

- [new 2.5] entry information: -e option

`rbh-report -e /mnt/lustre/dir1/file.1`

```
id             :      [0x200000400:0x16a94:0x0]
parent_id      :      [0x200000007:0x1:0x0]
name           :      file.1
path updt      :      2013/10/30 10:25:30
path           :      /mnt/lustre/dir1/file.1
depth          :      1
user           :      root
group          :      root
size           :      1.42 MB
spc_used       :      1.42 MB
creation       :      2013/10/30 10:07:17
last_access    :      2013/10/30 10:15:28
last_mod       :      2013/10/30 10:10:52
last_archive   :      2013/10/30 10:13:34
type           :      file
mode           :      rw-r--r--
nlink          :      1
status         :      modified
md updt        :      2013/10/30 10:25:30
stripe_cnt, stripe_size,      pool:           2,     1.00 MB,
stripes        :      ost#1: 30515, ost#0: 30520
```

- fileclasses summary: --class-info option

Once you have defined fileclasses (see next sections of this tutorial), you can get file repartition by fileclass:

`rbh–backup-report --class-info`

```
Class:       important_files
```

```
Nb entries:              879
Space used:           9.00 GB    (18874368 blks)
Size min:                0       (0 bytes)
Size max:             1.00 GB    (19327352832 bytes)
Size avg:            10.48 MB    (10993664 bytes)

Class:             project_B
Nb entries:              878
Space used:           2.23 GB    (4666481 blks)
Size min:                0       (0 bytes)
Size max:            95.37 MB    (100000000 bytes)
Size avg:             2.60 MB    (2727157 bytes)
```

- …and more:
  you can also generate reports, or dump files per directory, per OST, etc…
  To get more details about available reports, run '`rbh-backup-report --help`'.

3.2 rbh-backup-find

'find' clone accessing robinhood database.
Example:

```
rbh-backup-find /mnt/lustre/dir -u root -size +32M -mtime +1h –ost 2 -ls
```

3.3 rbh-backup-du

'du' clone acessing robinhood database. It provides extra features like filtering on a given user, group or type…
Example:

```
rbh-backup-du –H -u foo /mnt/lustre/dir.3

45.0G   /mnt/lustre/dir.3
```

4 Archiving

Now we know how to setup and query robinhood, let's archive the filesystem data.
Robinhood archives data incrementally. In other words, it only copies new or modified files but do not copy unchanged files multiple times.
It can perform multiple copies in parallel (possibly using several client nodes).
Admin can set the priority criteria that determines the copy order: it can be last modification time, last archive time, creation time, last access time… By default, copy priority is based on last modification time (oldest first).

4.1 Using a single default policy

Robinhood makes it possible to define different migration policies for several file classes.
In this example, we will only define a single policy for all files.
This is done in the 'migration_policies' section of the config file:

```
migration_policies {
      Policy default    {
            condition { last_mod > 1h }
```

```
        }
}
```

'default' policy is a special policy that applies to files that don't match a file class.
In a policy, you must specify a condition for allowing entries to be migrated. In this example, we don't want to copy recently modified entries (modified within the last hour).

Run '`rbh-backup --migrate --once`' to apply this policy once.
You can also run it as a daemon (without the '--once' option). It this case, it will periodically run the migration on eligible entries

4.2 Defining file classes

Robinhood makes it possible to apply different migration policies to files, depending on their properties (path, posix attributes, …). This can be done by defining file classes that will be addressed in policies.

In this section of the tutorial, we will define 3 classes and apply different policies to them:

- We don't want "*.log" files that belong to 'root' to be archived;
- We want to quickly archive files from directory '/mnt/lustre/saveme' (1hour after their creation, then backup hourly as long as they are modified).
- Archive other entries 6h after their last modification.

First, we need to define those file classes, in the 'filesets' section of the configuration file. We associate a custom name to each FileClass, and specify the definition of the class:

```
Filesets {
      # log files owned by root
      FileClass root_log_files {
            Definition  {
                  owner == root
                  and
                  name == "*.log"
            }
      }

      # files in filesystem tree /mnt/lustre/saveme
      FileClass saveme  {
            Definition  { tree == "/mnt/lustre/saveme" }
      }

}
```

Then, those classes can be used in policies:

- entries can be ignored for the policy, using a 'ignore_fileclass' statement;
- they can be targeted in a policy, using a 'target_fileclass' directive.

```
migration_policies {
      # don't archive log files of 'root'
      ignore_fileclass = root_log_file;

      # quickly archive files in saveme
      policy copy_saveme
      {
```

```
            target_fileclass = saveme;
            # last_archive == 0 means "never archived"
            condition { (last_archive == 0 and creation > 1h)
                        or last_archive > 1h }
      }

      # The default policy applies to all other files
      policy default {
            condition { last_mod > 6h }
      }
}
```

Notes:

- A given FileClass cannot be targeted simultaneously in several migration policies;
- policies are matched in the order they appear in the configuration file. In particular, if 2 policy targets overlap, the first matching policy will be used;
- You can directly ignore entries by specifying a condition in the 'migration_policies' section (without fileclass definition), using a 'ignore' block:

```
migration_policies {
   ignore { owner == root and name == "*.log" }
   …
```

A FileClass can be defined as the union or the intersection of other FileClasses. To do so, use the special keywords "union", "inter" and "not" in the fileclass definition:

```
FileClass root_log_A {
      Definition {
            (root_log_files inter A_files)
            union (not B_files)
      }
}
```

4.3 Migration parameters

Robinhood provides a fine control of migration streams: number of simultaneous copies, runtime interval, max volume or file count to be copied per run, priority criteria…
Those parameters are set in the 'migration_parameters' section. See the main parameters below:

```
Migration_Parameters {
    # simultaneous copies
    nb_threads_migration   = 4 ;

    # sort order for applying migration policy
    # can be one of: last_mod, last_access, creation, last_archive
    lru_sort_attr          = last_mod ;

    # interval for running migrations
    runtime_interval       = 15min ;
    # maximum number of migration requests per pass (0: unlimited)
    max_migration_count    = 50000 ;
    # maximum volume of migration requests per pass (0: unlimited)
    max_migration_volume   = 10TB ;
```

```
    # stop current migration if 50% of copies fail
    #(after at least 100 errors)
    suspend_error_pct = 50% ;
    suspend_error_min = 100 ;
}
```

4.4 Manual migration actions

In previous migration commands we run, all files were considered.
You can apply policies only to a subset of files:

- To apply migration policies only for files on a specific **OST**, run (one-shot command):
  `rbh-backup --migrate-ost=<ost_index>`
- To apply migration policies only for a given **user**, run (one-shot command):
  `rbh-backup --migrate-user=<username>`
- To apply migration policies only for a given **group**, run (one-shot command):
  `rbh-backup --migrate-group=<groupname>`
- To apply migration policies only for a given fileclass, run (one-shot command):
  `rbh-backup --migrate-class=<fileclass>`
- To apply migration policies to a single file, run (one-shot command):
  `rbh-backup --migrate-file=<filepath>`
- To force archiving all files and ignore policy time conditions, run:
  `rbh-backup --sync`

4.5 Customized copy command and remote copy agents

By default, robinhood uses a built-in function to copy data from the filesystem to the backend. You may need to perform specific actions for archiving, or use your favorite copy command instead.
To customize it, you can specify a copy command in the *Backend* block of the configuration file, using the 'action_cmd' parameter, e.g.:

```
action_cmd = /usr/local/bin/my_copy_wrapper.sh
```

The copy wrapper must implement the following interface:

cmd ARCHIVE <source_path> <target_path>
cmd RESTORE <source_path> <target_path>

A template script is installed with robinhood-backup: `rbhext_tool`.

You can use multiple copy nodes using `rbhext_tool_clnt` as copy command and `rbhext_tool_svr` as xinetd server on remote copy nodes.

4.6 Advanced backend parameters

The following parameters can be set in 'backend' config block:

- **archive_symlinks** (default = true): indicates if symlinks must be archived to backend filesystem. Set it to false if you are only interested in file's data, or to limit the amount of metadata in the archive. Note: in all cases, robinhood saves symlink information in its database so you can restore filesystem symlinks even if you don't archive them.
- **check_mounted** (default = true): By default, robinhood checks the specified path for the backend is mounted and matches the specified FS type. Disable this parameter to skip this check.
- **xattr_support** (default = false): indicates if the backend filesystem supports extended attributes. It is not used in current robinhood version.
- **compress** (default = false): compress files in the archive backend (gzip format).
- **sendfile** (default = false): the `sendfile' boolean parameter controls whether to use the optimized `sendfile(2)`-based copy mode. This can significantly improve performance but requires both following conditions to be met:
  - the destination filesystem must support fallocate().
  - the kernel must allow `sendfile(2)` on two regular files (linux kernel >= 2.6.33).
- **sync_archive_data** (default = true): force synchronizing data to disk when a file is written to the backend. This ensure data is saved immediately, but may impact performances for small files.
- **copy_timeout** (default = 6h): time before cleaning a temporary copy file in the backend.

5 Orphan cleaning and un-delete

5.1 Orphan cleaning in archive

Robinhood keeps track of deleted files in Lustre and clean the related entries in the archive after a certain delay (default delay is 1day).
This cleaning is done when running ‘rbh-backup --hsm-remove’ (with --once to run just one-shot).
To change the delay, or disable orphan cleaning, define a ‘hsm_remove_policy’ block in the config:

```
hsm_remove_policy {
    # set this parameter to 'off' for disabling removal in the archive
    hsm_remove = enabled;

    # delay before cleaning deleted object in the archive
    deferred_remove_delay = 30d;
}
```

5.2 Undelete

If *hsm_remove* is enabled, robinhood keeps track of deleted files and their original location during the specified *deferred_remove_delay*. During this delay, you can recover a deleted file from the archive, using ‘`rbh-backup-undo-rm`’ command.
For example, you can list all removed files from a given directory:

```
rbh-backup-undo-rm –L /mnt/lustre/my/lost/dir
```

and then recover its content:

```
rbh-backup-undo-rm –R /mnt/lustre/my/lost/dir
```

6 Daemon mode and ‘robinhood-backup’ service

So far, we only started actions independently using options like ‘--readlog’, ‘--migrate’, etc.
Note that can combine actions on the command line, e.g.:

```
rbh-backup --migrate --hsm-remove.
```

You can also run all actions in a single instance of robinhood running as a daemon, just by executing ‘`rbh-backup`’ with no argument.
This is also done by default when starting the ‘robinhood-backup’ service:

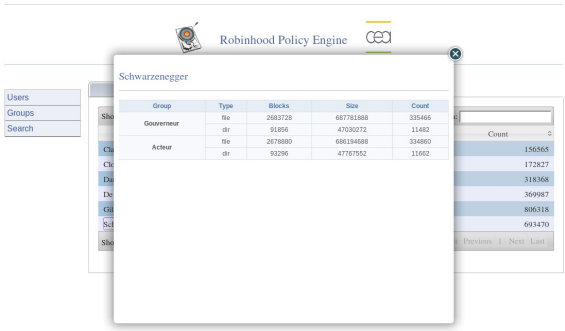
```
service robinhood-backup start
```

You can possibly change the behavior of service robinhood-backup, by editing `/etc/sysconfig/robinhood-backup`
For example:

```
RBH_OPT=”--readlog --–migrate”
```

7 Setting up web interface

Web interface makes it possible for an administrator to visualize top disk space consumers (per user or per group), top inode consumers with fancy charts, details for each user. It also makes it possible to search for specific entries in the filesystem.



You can install it on any machine with a web server (not necessarily the robinhood or the database node). Of course, the web server must be able to contact the Robinhood database.

Requirements: php/mysql support must be enabled in the web server configuration.
The following packages must be installed on the web server: `php, php-mysql, php-xml, php-pdo, php-gd`

The following parameter must be set in httpd.conf:

```
AllowOverride All
```

Install robinhood interface:

- install robinhood-webgui RPM on the web server (it will install php files into /var/www/html/robinhood)

or

- untar the robinhood-webgui tarball in your web server root directory (e.g. /var/www/http)

Configuration:

In a web browser, enter the robinhood URL: http://*yourserver*/robinhood
The first time you connect to this address, fill-in database parameters (host, login, password, …):

Those parameters are saved in: `/var/www/http/robinhood/app/config/database.xml`

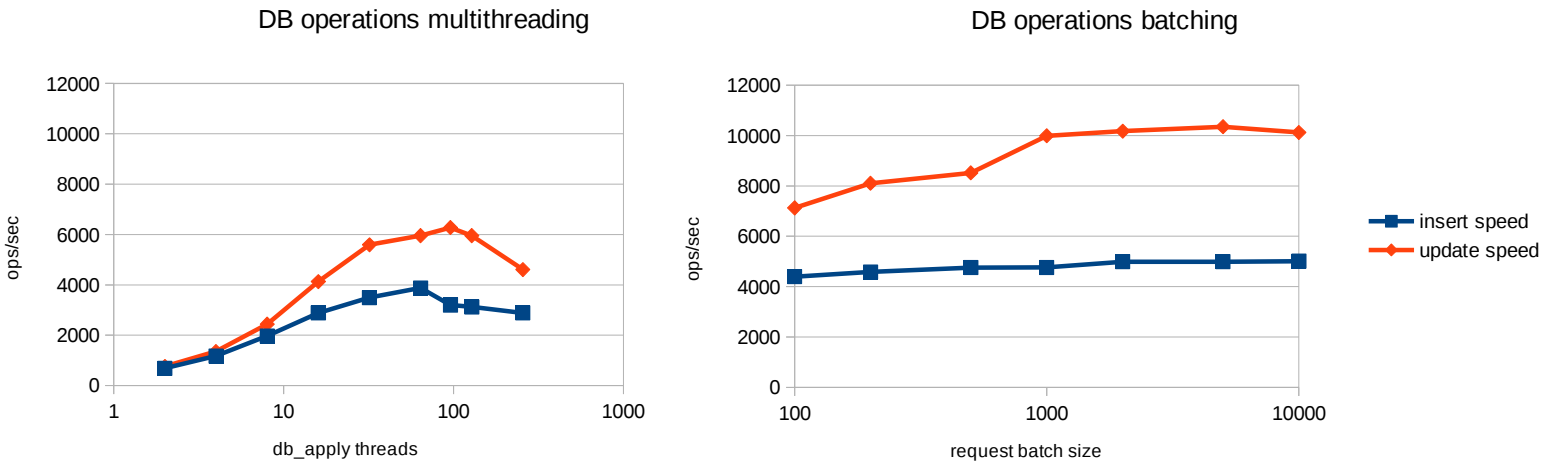
That's done. You can enjoy statistics charts.

8 Optimizations and compatibility

8.1 [new 2.5] Performance strategy for DB operations

You have the choice between 2 strategies to maximize robinhood processing speed:
- *multi-threading*: perform multiple DB operations in parallel as independent transactions.
- *batching*: batch database operations (insert, update...) into a single transaction, which minimize the need for IOPS on the database backend. Batches are not executed in parallel as they may results in DB deadlocks.

The following benchmark evaluates the DB performance for each strategy. This benchmark ran on a simple test-bed using basic SATA disk as DB storage for innodb.



The results show that batching is more efficient than multi-threading whatever the thread count, so it has been made the default behavior for robinhood 2.5.

You can control batches size defining this parameter in *EntryProcessor* configuration block (refer to the *admin guide* for more details about this block):

- **[new 2.5] max_batch_size** (positive integer): by default, the entry processor tries to batch similar database operations to speed them. This can be controlled by the max_batch_size parameter. The default max batch size is 1000.

If your DB storage backend is efficient enough (high IOPS) you may want to give a try to the other strategy. To switch from *batching* strategy to *multi-threading*, set **max_batch_size = 1**. This will automatically disable batching and enables multi-threading for DB operations. Consider increasing **nb_threads** parameter in this case (*EntryProcessor* configuration block):

- **nb_threads** (integer): total number of threads for performing pipeline tasks. Default is 4. Consider increasing it if you disable *batching*.

8.2 Database tunings

You can modify those parameters in `/etc/my.cnf` to speed-up database requests:
(tuning `innodb_buffer_pool_size` is strongly recommended)

```
innodb_file_per_table

# 50% to 90% of the physical memory
innodb_buffer_pool_size=16G

# 2*nbr_cpu_cores
innodb_thread_concurrency=32

# memory cache tuning
innodb_max_dirty_pages_pct=15

# robinhood is massively multithreaded: set enough connections
# for its threads, and its multiple instances
max_connections=256

# If you get DB connection failures, increase this parameter:
connect_timeout=60

# This parameter appears to have a significant impact on performances:
# see this article to tune it appropriately:
#http://www.mysqlperformanceblog.com/2008/11/21/how-to-calculate-a-good-innodb-log-file-size
innodb_log_file_size=500M
```

To manage transactions efficiently, innodb needs a storage backend with high IOPS performances. You can monitor you disk stress by running "`sar -d`" on your DB storage device: if '%util' field is close to 100%, your database rate is limited by disk IOPS.
In this case you have the choice between these 2 solutions, depending on how critical is your robinhood DB content:

- Safe (needs specific hardware): put your DB on a SSD device, or use a write-back capable storage that is protected against power-failures. In this case, no DB operation can be lost.
- Cheap (and unsafe): add this tuning to `/etc/my.cnf`:
  `innodb_flush_log_at_trx_commit=2`
  This results in flushing transactions to disk only every second, which dramatically reduce the required IO rate. The risk is to loose the last second of recorded information if the DB host crashes. This is affordable if you use to scan your filesystem (the missing information will be added to the database during the next scan). If you read Lustre changelogs, then you will need to scan your filesystem after a DB server failure.

This little script is also very convenient to analyze your database performance and it often suggests relevant tunings:

http://mysqltuner.pl

8.3 Optimize scanning vs reporting speed

By default, robinhood is optimized for speeding up common accounting reports (by user, by group, …), but this can slow database operations during filesystem scans. If you only need specific reports, you can disable some parameters to make scan faster.
For instance, if you only need usage reports by user, you had better disable *group_acct* parameter; this will improve scan performance. In this case, reports on groups will still be available, but their generation will be slower: if you request a *group-info* report and if *group_acct* is *off*, the program will iterate through all the entries (complexity: O(n) with n=number of entries). If *group_acct* is *on*, robinhood will directly access the data in its accounting table, which is quite instantaneous (complexity: O(1)).
Performance example: with *group_acct* parameter activated, *group-info* report is generated in 0.01sec for 1M entries. If *group_acct* is disabled, the same report takes about 10sec.

8.4 SLES init depedencies

On SLES systems, the default dependency for boot scheduling is on "mysql" service. However, in many cases, it should be too early for starting robinhood daemon, especially if the filesystem it manages is not yet mounted. In such case, you have to modify the following lines in `scripts/robinhood.init.sles.in` before you run `./configure`:

```
# Required-Start:    <required service>
```

8.5 Lustre troubles

- So far, it was recommended to disable Lustre `stat_ahead` on the robinhood host (caused Lustre crashes). Set this if your experiment crashed on the robinhood host.
  `lctl set_param llite.*.statahead_max=0`

- CPU overload and client performance drop when free memory is low (bug #17282): in this case, lru_size must be set at CPU_count * 100:
  `lctl set_param ldlm.namespaces.*.lru_size=800`

9 Getting more information

In-line help
‘rbh-backup --help’ and ‘rbh-backup-report --help’ provide a detailed descriptions of command-line parameters.
‘rbh-backup --template’ generates a documented configuration template that can help you for writing your configuration file.

Admin guide
For more details about robinhood configuration parameters, you can refer to the admin guide.
It is available in the download section of the robinhood project on sourceforge.
It is also located in the `doc/admin_guides` directory of the distribution tarball.

Project website and wiki
You can get extra information on the website, especially in the wiki: http://robinhood.sf.net

Mailing lists

- You can get support by posting your issue to robinhood-support mailing list: robinhood-support@lists.sourceforge.net

- To keep updated about new versions of robinhood, subscribe to: robinhood-news@lists.sourceforge.net

- Robinhood is Open Source. To contribute to its development, join: robinhood-devel@lists.sourceforge.net