Using Robinhood to Purge Data from Lustre File Systems

Tina M Declerck
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA USA
TMDeclerck@lbl.gov

***Abstract*— NERSC purges local scratch file systems to ensure end user usability and availability along with file system reliability. This is accomplished through quotas, and by destructively purging files that are older than a specified period. Prior large scale systems at our site utilized Lustre file systems which were comprised of servers directly connected to Lustre storage nodes. As such, the site had full access to the Lustre storage nodes allowing us to develop programs using the tools and libraries within the Lustre environment/infrastructure. Our latest file systems are utilizing Cray's Sonexion Lustre storage nodes which limit access to the underlying Lustre environment as they are designed to be appliance-like. Since existing APIs and libraries were no longer accessible we were required to develop new purge mechanisms. To address this Cray has provided Robinhood, which collects file system metadata into a database that can be queried for either data or metadata operations without having adverse affects on file system performance. This paper will describe in detail how our new purge mechanism was developed and deployed based upon the Robinhood capabilities. In general, similar requirements exist across sites with regard to identifying files to purge but this paper will cover those requirements along with those specific to NERSC's environment. A detailed description of the Robinhood policy engine – its capabilities, and how it is used at our site for purging will be the primary focus of the paper. In addition, the tools we use to tune the data obtained and ensure the appropriate files are being purged will be highlighted. The actual purge operation is a separate step to ensure data and metadata are consistent before a destructive purge operation takes place since the state of the file system may have changed during Robinhood's sampling period. NERSC's defined process for the purge operation will be thoroughly described. As part of the purge mechanism a list of purged files is then provided to the user in the root of their scratch directory. Other details of the purge will also be included such as how long the purge takes, an analysis of the data being purged and it's affect on overall process, as well as work done to improve the time required to purge. Finally, a discussion of the issues that we encountered and what was accomplished to resolve them.**

Keywords-robinhood; purge; Lustre;

I. INTRODUCTION

The National Energy Research Scientific Computing Center (NERSC), located at Lawrence Berkeley National Laboratory (LBNL), is the production scientific computing facility for the United States Department of Energy (DOE) Office of Science. NERSC provides computational resources to support the scientific activities of over 5000 users worldwide, resulting in huge demands for resources and storage. Over time file systems tend to fill making them nearly unusable. Even a nearly full file system can have an impact on performance and usability. To ensure the scratch file systems at NERSC can maintain reasonable performance, we use both user based disk quotas and regular purging to maintain the file systems at a reasonable level. Although NERSC has been purging for many years, the appliance-like Cray Sonexion file systems, delivered with our Cray XC-30, Edison, don't allow direct access to the Lustre servers making the existing purging method unusable. To remedy this, Cray provided a tool that uses the Robinhood Policy Engine. NERSC's specific requirements make direct use of Robinhood inadequate, so additional work was required. This paper will describe our requirements and how they are met using both Robinhood and local code to provide the purge capability. In addition, it will describe the Robinhood Policy Engine, and some of the issues we have encountered as well as any workarounds or fixes and additional capabilities available with Robinhood.

II. REQUIREMENTS

NERSC has had a purge capability for many years, however the storage solution provided for Edison, the Cray Sonexions, prevent the existing solution from working. Previously the Lustre file systems were configured on standard servers with access to disk arrays. We therefore had access to the Lustre metadata server (MDS) node. Purging on these file systems is done with a local C program using the low level file system libraries with a tool called ne2scan. With the appliance-like Cray Sonexion file systems, direct access to Lustre's MDS isn't available.

The minimum requirements for purge were provided to Cray in the statement of work when Edison was purchased.

It required the purge capability to enable automated file selection based on specified criteria and complete the unlink operation on the identified files daily. The unlink rate should be 7,500,000 per hour for files one byte to just under the file system block size and directories. In addition, the ability to select files based on the following attributes for all files in the file system: file size, last status change time (ctime), last modified time (mtime), last access time (atime), owner, group, and pathname.

Additional considerations based on lessons learned over years of purging add more requirements and help provide an understanding of the configuration used and the decisions made. Most sites understand the need to purge, but what are the reasons for what gets purged and how purging is done? Over time, purge policies at NERSC have changed for various reasons. For ease of maintenance, an attempt at keeping configurations between systems consistent is also a consideration. These policies are in these primary areas:

- exclusions
- what to purge
- how to purge
- purge policy

Each of these will be described briefly.

First, providing the ability to exclude users, groups, or specific directories is self-explanatory. However, initially user and group names were provided and in some cases only UID and GID were available so those users or group files ended up getting purged even though the user or group name were in the exclude lists. Now only the UID and GID are used for exclusion lists. Directory exclusions use a full path since multiple file systems can make it unclear otherwise.

Next, what files to purge appears on first glance to be evident. In general, it is. Accidental purging of files due to some special cases resulted in a couple of changes. For instance, early on an issue came up when a user had pulled a tar file from storage and untarred it preserving the original timestamps resulting in those files getting purged in the next purge cycle. That was clearly not the behavior the user expected or wanted. Since other users could do the same thing a solution was needed. That lead to SAFEDAYS – a parameter that is used to preserve files for the number of days specified if the ctime, which did get updated, was newer than the access or modify times. The second issue resulted in not removing directories. Some of our users will create a directory structure that is used by all their jobs. Unfortunately, the users do not necessarily verify the existence of the directory prior to use, so if the directory didn't exist their jobs would fail. This works well with Robinhoods default policy that also does not remove directories.

How to purge refers to a hard requirement to verify timestamps prior to purging. In the past the file system scan took hours to complete so there were cases when a user's files were purged even though they had changed. With Robinhood, since the data is generated from a database the purge data list generation has been relatively fast. But there have been cases when the data is not current either because it is not up-to-date with the file system changelogs or something is not working correctly, so it is still necessary to verify the data prior to purging even though data collection is much faster. This document will provide numbers showing how many files have not met the criteria for purge after the list was generated in section VII.

Finally, what is the basis for the purge policy? Our site has chosen to purge on a daily basis to a specified age. This provides our users with a known timeframe allowing them to plan their data needs appropriately (of course this does lead to a small problem with 'touch' where users touch files to try to keep them from getting purged). Other options have been to allow a file system to fill to something like 80% and then purge, or to purge the users who are using the most space. In the first case, the users became frustrated with the uncertainty of file age before purging and could lead to data that was needed getting purged. The second case is less fair and punished the users who used the system the most.

One additional feature for our users is a list of files purged placed in the root of the purged directory. This provides a way for users to know what files were removed and mostly eliminates calls to User Services about files that disappeared.

III. ROBINHOOD POLICY ENGINE

The Robinhood Policy Engine was developed by the Commissariat a l'energie atomique et aux energies alternatives / a Direction des applications militaries (CEA/DAM). It is a tool that provides the ability to manage a file system. It uses a MySQL database and tools to scan a file system and put the relevant information into the database. Although Robinhood has the ability to manage different file system types, this paper will focus on Lustre. Once the data is in the database it can be used for various activities like purging, reporting, auditing, accounting, etc. Robinhood also provides the ability to provide alerts for various things such as a file system filling up. Robinhood was specifically developed for high performance computing with an expectation of large file systems with lots of data and transactions. It performs its tasks in parallel for high performance.

One of the features in Lustre, available after version 2.0, is a changelog that allows other "users" to see a log of changes to files in the file system. With this capability, the Robinhood database can be kept up-to-date by reading the changelog and updating the database accordingly. This means the high overhead required to scan the file systems is not required. There are some potential issues with getting this working if care isn't taken. This will be covered in the configuration section. Robinhood also has other special capabilities for Lustre. It can track the usage by OST and purge a specific OST. It can list files per OST and policy criteria can be based on pools and OST index.

Robinhood's policy engine has many capabilities to purge using different options to specify when and how purging is done which are defined in a configuration file. Each file system has a separate database with it's own configuration defined. The configuration is defined in sections by keyword. Within each section a keyword=value

pair is used to provide the requirements. For instance, to set a purge to occur only when a file system is up to 80% full and purge down to 60% full, specifying the check occur every 6 hours (note the term "global_usage" is a keyword other options include user_usage, group_usage, and periodic):

```
Purge_Trigger
{
        trigger_on              = global_usage ;
        high_threshold_pct      = 80% ;
        low_threshold_pct       = 60% ;
        check_interval          = 6h ;
}
```

Ignoring specific users, directories, or files can also be specified in a purge policy as well as identifying specifics such as the required age of a file before it can be purged. Directories have a separate definition section and can be ignored or purged if they are empty for a specified period of time. Alerts can be set up to go to an alerts log and also emailed. Email alerts can be batched in order to reduce the spam factor.

Robinhood provides some nice tools for accessing the database. These include rbh-find, rbh-du, rbh-diff, and rbh-report. rbh-find is a find like tool that can be used to access files based on name, age, type, etc. rbh-du works like du. rbh-diff can compare the data in the database with a scan of the file system and can optionally either update the database or revert the file system to match the database. rbh-report has some pre-configured report options to provide information about the files based on user, group, top-users, etc. For example, this report shows a basic breakdown of the file system:

```
# rbh-report -f scratch1 -i
Using config file '/etc/robinhood.d/tmpfs/scratch1.conf'.
type,     count, volume,   min_size,   max_size, avg_size
symlink, 470413, 33.30 MB,        2,        236,       74
dir,    1565317,  9.19 GB,  4.00 KB,   53.54 MB,  6.16 KB
file,  25063588, 291.37 TB,       0,    8.58 TB, 12.19 MB
fifo,       40          0,        0,          0,        0

Total: 27099727 entries, 320378342825978 bytes (291.38
TB)
```

More specifics about Robinhood can be found in the tutorial[2] and administrators manual[1]

IV. System Configuration

A. File system Configuration

NERSC's Edison system consists of three separate Cray Sonexion 1600 Lustre file systems. Two file systems, scratch1 and scratch2, are each configured with 12 Scalable Storage Units (SSU) for a total usable capacity of 2.16 PB each. These file systems are used for standard user scratch data with the users distributed across them. The third file system, scratch3, consists of 18 SSUs with a total usable capacity of 3.24 PB. This file system services special requests for users with a requirement for higher bandwidth applications. Fine grain routing is used for performance to the Cray; a flat network is used for access by the login nodes. More specifics on the file systems can be found in Table I. Each Object Storage Target (OST) is comprised of an 8x2 RAID6 with 3TB near-line serial attached SCSI (SAS) disks.

B. Purge Server

The purge server is a stand-alone server. It is configured with:
4 x 8-core Sandy Bridge @ 2.40 GHz
512 GB memory
2 x 900 GB 10k disks

Operating System is SLES11 SP2.
Lustre client 2.2

TABLE I. NERSC SCRATCH FILE SYSTEM LAYOUT

	scratch1 & scratch2	**scratch3**
2 OSS / SSU	24 OSSs	36 OSSs
4 OST/ OSS	96 OSTs	144 OSTs

C. Other Information

The quotas, which are enabled but not enforced, are set to 5M inodes and 10TiB per user on scratch1 and scratch2. Quotas are also enabled but not enforced on scratch3 but access is currently unlimited. Although not enforced via the quota configuration, the quotas are maintained by verification at job submit time and in the job prologue. If a user is over quota they are not allowed to submit jobs.

V. Robinhood Set-Up & Configuration

A. Installing Robinhood

The Robinhood Policy Engine First Steps Tutorial manual[2] describes the installation relatively well. The simple version is download, build, install, then configure. The configuration consists of creating the database, creating a configuration file and then running Robinhood. However, a few issues hindered the initial install of Robinhood. These have, for the most part, been resolved by some bug fixes to the code by Cray. Additionally, getting the best performance out of MySQL can be challenging. MySQL provides many configuration options that can be tweaked but figuring out which ones to tweak and how much can be difficult. In addition, there were problems with versions of MySQL prior to 5.5. Robinhood also provides some tuning around the number of threads for each stage of processing. The options used on Edison are documented here and may provide a starting point in establishing configuration parameters should your site decide to try to use Robinhood.

Security may be a concern for some sites. The purge server on Edison has limited access, so is less a concern at our site. The database has information about files and directories that would not be typically accessible to a normal user. In addition, because the data on the file system must be accessible by the Robinhood Policy Engine, Robinhood must

run as root. The purge server on Edison only collects data; it is not used for purging, so some of the potential risk is mitigated.

Requirements:
MySQL 5.5 (recommended)
Robinhood 2.5.x
    robinhood-recov-tools
    robinhood-tmpfs
    robinhood-adm

Of note, MySQL version 5.5 provided better results than previous versions. Robinhood version 2.5.0-0.alpha1 is currently installed on Edison, which has fixes provided by Cray for some issues we ran into early on in testing. The current version available is 2.5.1.

Table II shows the mySQL options configured to improve database performance in the [mysql] portion of the /etc/my.cnf configuration file. For more information on these options see:

http://docs.oracle.com/cd/E19957-01/mysql-refman-5.5/index.html

In addition, to allow mysql to use large pages the following parameters in the /etc/security/limits.conf required modification:

```
mysql hard memlock unlimited
mysql soft memlock unlimited
```

Parameters in /etc/sysctl.conf also needed tuning:

```
kernel.shmmax=50000000000
kernel.shmall=32534377267
vm.nr_hugepages = 50000 # 25000*2MB=50G
vm.hugetlb_shm_group = 202 # mysql group
```

The Robinhood configuration file is mostly self-explanatory. The stats_interval was set to 1 min so the options used could be verified by reviewing the logs during scanning. The following shows some of the options that may not be as easy to figure out from the documentation.

```
ListManager {
        commit_behavior = transaction;
        user_acct = disabled;
        group_acct = disabled;
        MySQL {
                innodb = enabled;
        }
}
FS_Scan {
        Nb_threads_scan = 32;
}
```

TABLE II. MYSQL PARAMETER CHANGES AND DESCRIPTION

`innodb_flush_log_at_trx_commit = 0`	Allows MySQL to batch updates. Although it also has the potential to lose up to 1 sec of transaction data. Options 0, 1 (default), and 2
`max_connections= 512`	Maximum permitted number of client connections (default 151)
`innodb_buffer_pool_size= 30G`	Memory where table and index data are cached. Set to ~50% of defined memory. NOTE: The boot option mem=70G is used on our purge server
`innodb_max_dirty_pages_pct= 15`	Defines the % of dirty pages before flushing. Default 75%.
`innodb_thread_concurrency= 32`	Keep os threads inside innoDB at or below this level (default unlimited)
`innodb_log_file_size= 100M`	Size in bytes of a log file in a log group (default 5M)
`innodb_log_buffer_size= 50M`	Size in bytes of the buffer used to write log files on disk (default 8M)
`innodb_data_file_path= ibdata1:1G:autoextend`	Paths to the individual data files and their size
`large-pages`	If available (must be turned on in the kernel) can increase performance due to fewer TLB (translation lookaside buffer) misses
`table-open-cache= 2000`	Number of open tables for all threads (increases required number of file descriptors)
`sort-buffer-size= 32M`	Size of buffer allocated to do sorts
`read-buffer-size= 16M`	Each thread that does a sequential scan allocates a buffer of this size for each table it scans
`read-rnd-buffer-size= 4M`	When reading rows in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks
`thread-cache-size= 128`	Number of threads the server should cache for reuse
`query-cache-size= 40M`	Amount of memory allocated for caching for query results
`query-cache-limit= 1M`	Do not cache results that are larger than this number of bytes (default 1MB)

`tmp-table-size= 16M`	max size of internal in-memory temporary tables.

```
EntryProcessor {
        Nb_threads = 24;
        Max_pending_operations = 8000;
}
```

The Robinhood logs have 2 sections that are logged for every interval – the "General statistics" and the "EntryProcessor Pipeline Stats". The log file location is specified in the Robinhood configuration file. While testing, check the statistics in the EntryProcessor Pipeline section for the "Wait" count in the stage 0 line. It should be large while scanning. Adjust the max_pending_operations and the nb_threads in the EntryProcessor section of the configuration file to improve performance.

The Lustre changelogs also need to be configured. Robinhood has a script that will configure this if the MDS and MGS are on the same host:

```
# rbh-config enable_chglogs
```

Then the changelog user (usually something like 'cl1') will need to be added to the Robinhood configuration file. If the MDS and MGS are NOT on the same host, this will need to be configured manually. The first step is to ensure all events are enabled. This can be checked on the MGS by running:

```
# lctl get_param mdd.*.changelog_mask
mdd.snx11111-MDT0000.changelog_mask= MARK
CREAT MKDIR HLINK SLINK MKNOD UNLNK
RMDIR RNMFM RNMTO OPEN IOCTL TRUNC SATTR
XATTR HSM MTIME CTIME
```

The default does not include a CLOSE event. If this is not in the list, run:

```
# lclt conf_param mdd.*.changelog_mask all-ATIME
```

Then rerun the get_param command. It should now show output like:

```
mdd.snx11111-MDT0000.changelog_mask=MARK
CREAT MKDIR HLINK SLINK MKNOD UNLNK
RMDIR RNMFM RNMTO OPEN CLOSE IOCTL
TRUNC SATTR XATTR HSM MTIME CTIME
```

Then on the MDS the changelog consumer needs to be registered. This requires a changelog reader id referred to in this document as the changelog user. To create the user run:

```
# lctl
lctl> device snx11111-MDT0000
lctl> changelog_register
snx11111-MDT0000: Registered changelog userid 'cl1'
```

If multiple readers are created without a process accessing and clearing it, or the reader id doesn't match the one in the Robinhood configuration file, the log will not get cleared which can cause problems. These will be described in more detail in the issues section. Therefore, it is important to verify the information on the MDS by running:

```
# lctl get_param mdd.*.changelog_users
    mdd.snx11111-MDT0000.changelog_users=current
index: 7958874742
    ID    index
    cl1   6411236842
```

Once the configuration is ready, start robinhood using the init script provided in the robinhood-tmpfs rpm. This will allow the changelog reader to start updating the database.

Although the Robinhood Policy Engine appears to be very capable, Edison only uses Robinhood's database and it's ability to obtain the changelog data from the Lustre file systems. Additional testing is currently underway to evaluate it's ability to keep state with the file systems before it's purge capabilities will be used in production.

VI. Operation

Since NERSC doesn't use all of the capabilities of the Robinhood Policy Engine, the process required some additional programming. This process consists of a manager perl program, purge_mgr.pl, which controls the flow of activities. The manager then utilizes additional programs, commands, and configuration files to create and and purge the files identified. Each of these steps will be described in detail.

Robinhood was used to perform an initial scan of each file system providing the base file system data for the Robinhood database, using the Robinhood configuration files in /etc/robinhood.d/tmpfs/<file system>.comf. The –f identifies the configuration file to use. If only one configuration file exists in the default location, /etc/robinhood.d/tmpfs/, that option is not necessary. The RBH_CFG_DEFAULT environment variable can also be used to identify the configuration file. The –scan is using the scan option and the –once says to only scan the file system one time. The process can also be daemonized using the –detach option.

```
# robinhood –f scratch1 –scan –once
```

The database is then kept in sync with the file system using the Lustre changelogs as described in the Robinhood Set-up and Configuration section. When the set up is complete the regular purge is done. The purge_mgr.pl program first reads the local configuration file. The purge configuration file is fairly simple using keyword, value pairs. It currently specifies how long data should be kept, PURGEDAYS, the previously mentioned SAFEDAYS, and any exclusion of directories, users, or groups. The PURGEDAYS and SAFEDAYS are assigned variables and the exclusions are placed in arrays.

The purge hit list is generated using the rbh-find command from the purge_mgr.pl program.

```
rbh-find -f scratch1 -lsstat /scratch1/scratchdirs -atime +84 >
\ /var/log/purge/scratch1.purge.<date>
```

The file system is a command line option, the atime opton is the PURGEDAYS from the configuration file and the output file is going to a default location. The location can be changed using a command line option if desired. The –lsstat option provides the stat information for each file:

```
[file,1234,5678,2287,1379445176,1379445176,1379445061
              ]=/scratch1/scratchdirs/user1/file1
  ^   ^      ^    ^         ^              ^           ^
  |   |      |    |         |              |           |
  |   |      |    |         |              |           +--- mtime
  |   |      |    |         |              +---- atime
  |   |      |    |         +---- ctime
  |   |      |    +--- size in bytes
  |   |      +--- gid
  |   +--- uid
  +--  file|dir|symlink|block|char|fifo|socket
```

The next step is to remove the excluded users, groups, and directories from the list, any files that fit the SAFEDAYS criteria, and any directories. This is accomplished by checking each file with the information in the arrays created when the local configuration file was read. If there is a match the next file is read and checked against the exclusions. If the file is ok to purge, it is added to a new file that is also stored in the /var/log/purge directory. When all files have been checked then the purge is run.

The purge operation is a python program provided by Cray. Based on NERSC's requirements, each file is checked to ensure any of the stat data match the data in the database prior to removing the file. This is a very good idea since there have been instances when the data on the file system and the database don't match. These appear to be the result of various configuration settings getting changed. However, without the additional verification, files would have been purged that should not have been. These will be described in more detail later. The purge program has threading capabilities and is run using the following options:

```
npurge.py --do-purge --num-threads=30 --log-
deleted=/var/log/purge/scratch1.purged.<date>
/var/log/purge/scratch1.purge<date>_1
```

The –do-purge says to actually purge versus a dry run. The –num-threads tells how many threads to use to perform the verification and purge, the –log-deleted will log the deleted file to the specified location and the final option is the list of files to purge. The output is also logged to track files that are not purged due to a mismatch during verification.

The final step of the purge operation is to add a file to each users scratch directory, .purged.<date>. This provides the user with information to know why their files may have disappeared. This has proved to be very helpful and has reduced calls to the customer services department.

Although Robinhood has worked fairly well, there have been a few problems. The server from where all of the Robinhood databases for the three file systems reside hung during a scan of all the file systems. The server couldn't be accessed and the server required a reboot. The scan had to be restarted but there was no other fallout. Scans are now only run on one file system at a time. The current configuration is working but being monitored to ensure higher loads on the file system won't result in issues with either the server or the database updates.

One of the file systems became inaccessible as the result of the changelog filling. Some of this may have occurred during an upgrade because the lctl set_param was used to configure the data instead of the more permanent lctl conf_param. Exactly what happened is uncertain. However, one of the lessons learned is that if the changelog user identified in the Robinhood configuration file does not match the user on Lustre, not only are the changelogs not available but they also will not be cleared. Although it may be possible to attempt to clear the changelog without causing problems, it was preferred that the file system is not being accessed while the problem is being addressed. This means that the changelog has to be disabled and the file system is no longer getting updates except through a scan. To be safe, until confidence in the state of these parameters is established, a verification of both the changelog user and the changelog mask has been added to the procedures after an update. This includes verifying the changelog user id matches what is in the Robinhood configuration file.

During the purge operation there are times when several files are not getting purged due to a mismatch in the verification process. Some of these issues with the database and the file system getting out of sync appear to be based on lack of knowledge regarding the changelog and the events that need to be generated. This is explained above. There has not been sufficient time with the correct settings to determine if there is an underlying problem or if this is all just a parameter setting issue.

One final issue was that the rbh-find would crash when trying to access one of the databases. It always occurred at the same point. The bug was found and the fix has been included in the Robinhood v2.5.

VII. STATISTICS

Following are some statistics for various tasks. With the issues with the changelog there have been several opportunities to get timing on file system scans both while the system is quiet and during normal use. Timing has been collected during scans, generating the purge hit list, and the purge operation.

The scan operation has always been performed on a clean database. Prior to starting the scan the changelog following is stopped. In all cases the database was then zeroed out and the file system was scanned using the following command sequence:

```
# rbh-config empty_db robinhood_scratch3
# robinhood -f scratch3 --scan –once
```

TABLE III. ROBINHOOD SCAN TIMING

Date	File System	Time (sec)
4/3/14	/scratch1	22445
4/2/14	/scratch2	8853
3/31/14	/scratch3	10258

The Table III shows is the time required for a complete rescan of the file systems. Once the scan completed and the changelog read restarted, it took about 3 ½ hours for the backlog of changelog entries to be processed. The read was started approximately 24 hours after the new scan of the file system.

Time to generate the purge list from the database is about 20 min. as shown in Table IV.

TABLE IV. PURGE LIST GENERATION TIME

Date	File system	Time
2/14/14	/scratch1	1101 sec
2/18/14	/scratch2	1159 sec

Purge time varied considerably based on activity. The purge operation was completed while there was normal use of the file systems. Table V provides information on the purge time, rate, number of files purged, and also the number of files that weren't purged due to a mismatch between the database and the file system. Changes were noticed for almost every field, ctime, atime, mtime, gid, and size. There was typically not a long period between the generation of the purge hit list and the purge operation so this seems to indicate the database not quite being in sync. See the issues in the previous section for more information.

TABLE V. PURGE TIME

Date	File system	Time (sec)	# Files	Rate (files/sec)	# Files Not Purged
2/14/14	/scratch1	4441	3,587,437	808	2054
3/30/14	/scratch1	4691	8,613,585	1836	124986
1/13/14	/scratch2	2682	5,130,053	1913	1566
2/18/14	/scratch2	5590	6,596,284	1180	3867
3/30/14	/scratch2	3183	4,891,038	1537	126586
3/31/14	/scratch3	177	87,423	494	1600

VIII. FUTURE WORK

The Robinhood database contains useful data. Collecting more statistics on filesystem usage may provide useful insights into better ways to handle user data.

ACKNOWLEDGMENT

Thank you to David McMillen and Frank Zago at Cray for the initial Robinhood configuration, python purge utility, npurge.py, and assistance with the various issues. Cray has also provided several improvements and fixes to the Robinhood developers.

This work was supported by the Director, Office of Science, Office of Advance Scientific Computing Research of the U.S. Department of Energy under contract No. DE-AC02-05CH11231.

REFERENCES

[1] Thomas Leibovici, "Robinhood PolicyEngine Admin Guide," V 2.5.0 Feb 12, 2014.

[2] Thomas Leibovici, "Robinhood PolicyEngine First Steps Tutorial," V 2.5.0 Feb 13, 2014

[3] Oracle , "MySQL 5.5 Reference Manual," 2010 sections 5.1, 7.5