Hierarchical Storage Management Configuration Guide

Implementation Guide
High Performance Data Division

US

August 12, 2015

World Wide Web: http://www.intel.com

Intel® Confidential

Disclaimer and legal information

Portions of the source code contained or described herein and all documents related to the source code ("Material") are owned by Intel® Corporation or its suppliers or licensors. Title to the Material remains with Intel® Corporation or its suppliers and licensors. The Material contains trade secrets and proprietary and confidential information of Intel® or its suppliers and licensors. The Material is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel® in writing.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL® ASSUMES NO LIABILITY WHATSOEVER AND INTEL® DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel® Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL® AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL® OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL® PRODUCT OR ANY OF ITS PARTS.

Before using any third party software referenced herein, please refer to the third party software provider's website for more information, including without limitation, information regarding the mitigation of potential security vulnerabilities in the third party software.

Intel® may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel® reserves these for Lustre Architecture future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel® literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm.

Intel® and the Intel® logo are trademarks of Intel® Corporation in the U.S. and/or other countries.

* Other names and brands may be claimed as the property of others.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)

Contents

About this Document

Document Purpose

This document describes how to configure a hierarchical storage management system for a Lustre file system created with Intel® Enterprise Edition for Lustre* and Intel® Manager for Lustre. This document introduces HSM, its functionality, and its various components. It then describes how, using Intel® Manager for Lustre, to:

1. Enable the HSM Coordinator process.
2. Provision an HSM Agent host.
3. Add the POSIX copytool instance to the HSM Agent host (and start copytool).

The document then discusses how to setup and configure the Robinhood HSM policy engine, which is bundled with Intel® EE for Lustre.

Intended Audience

This guide is intended for systems integrators with a strong technical background in Linux system administration as well as Lustre file system deployment and management, and who have a requirement to support access to Lustre from Windows clients. The guide tries to make no assumptions about the reader's experience HSM, but advanced concepts may require some knowledge of Windows network services such as Active Directory or Windows NT4.

It is expected that readers have:

- experience administering file systems and storage infrastructure, and familiarity with storage concepts such as RAID, SAN, and LVM
- system management experience sufficient to install and configure a storage platform compatible with the requirements as defined in this guide
- proficiency in setting up, administering, and maintaining computer networks, including Ethernet, TCP/IP and InfiniBand where necessary. Some knowledge of Lustre networking (LNET) is required.
- some experience with installing and managing Lustre file systems

This document is not intended for end-users or application developers.

Conventions Used

Conventions used in this document include:

- # preceding a command indicates the command is to be entered as root.
- $ indicates a command is to be entered as a user.
- *<variable_name>* indicates the placeholder text that appears between the angle brackets is to be replaced with an appropriate value.

Introduction

A Lustre* file system is a network-based, distributed parallel storage platform consisting of metadata servers (MDS) and object storage servers (OSS). Metadata servers manage the file system namespace (directory structure, file information – inodes), while object storage servers contain the file contents. A management server (MGS) provides directory services, storing the configuration information for Lustre file systems and presenting that information to the population of Lustre servers and clients.

Intel® Manager for Lustre software is an additional service supplied with Intel® Enterprise Edition for Lustre that enables administrators to create, manage and monitor Lustre file systems. With this software, operators can configure and manage servers, as well as monitor file system health and performance.

Figure 1 depicts the minimum configuration of an enterprise deployment of Lustre. The metadata servers and object storage servers are configured in high availability (HA) cluster pairs using a standard template based on Pacemaker and Corosync. Each pair has equal access to shared external storage. Each pair shares a dedicated heartbeat network interconnect (provided through a cross-link cable) and has access to a common management network for communicating with the Intel® Manager for Lustre software and for cluster-management communications. Lustre file system traffic is sent over a high-performance network, typically InfiniBand or 10Gb Ethernet.

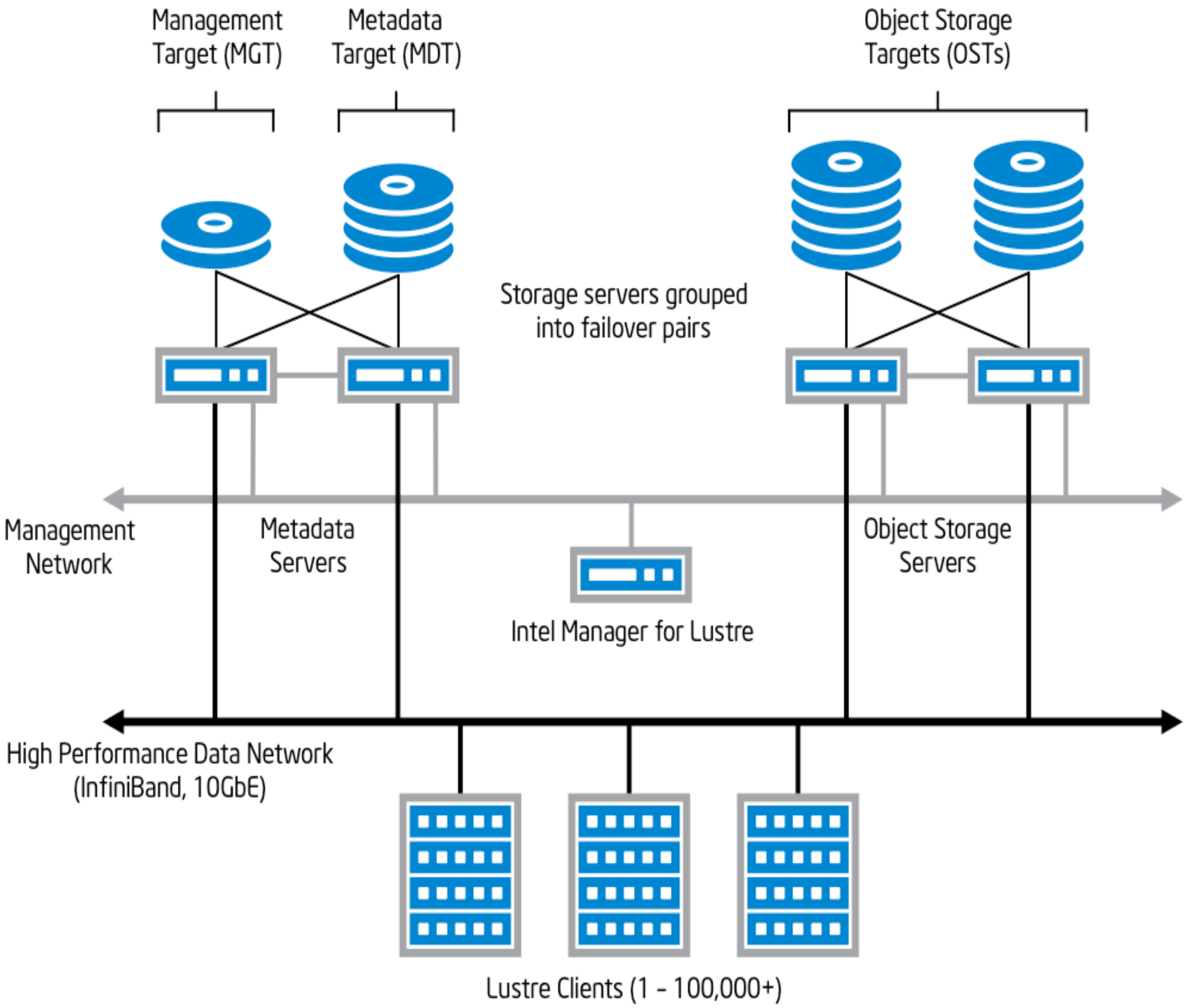

Management Target (MGT)
Metadata Target (MDT)
Object Storage Targets (OSTs)
Storage servers grouped into failover pairs
Management Network
Metadata Servers
Object Storage Servers
Intel Manager for Lustre
High Performance Data Network (InfiniBand, 10GbE)
Lustre Clients (1 - 100,000+)

Figure 1. Intel® Enterprise Edition of Lustre* architecture

An HA storage server pair represents a single scalable unit in the Lustre file system. To expand the capacity and throughput performance of the file system, add more OSS pairs. To improve metadata performance, add more MDS pairs. This capability is provided by Lustre's Distribute Namespace technology (DNE). (DNE is not available in the 1.0.x release of Intel® EE for Lustre.) Storage servers can be added to the file system at any time without interrupting service to the client population. By using an HA storage server pair as the scalable unit, the file system can be safely expanded without compromising service and data availability to the client population. In order to use Intel® Manager for Lustre with Intel® Enterprise Edition for Lustre, high-availability hardware configurations for the metadata servers and object storage servers are required. Each server in an HA server pair must have a near-identical hardware and software configuration.

For a comprehensive description of the system configuration requirements for Intel® Enterprise Edition for Lustre, refer to the Intel® Enterprise Edition for Lustre Partner Installation Guide.

Hierarchical Storage Management Architecture

Hierarchical Storage Management (HSM) is a collection of technologies and processes designed to provide a cost-effective storage platform that balances performance and capacity. Storage systems are organized into tiers, where the highest-performance tier is on the shortest path to the systems where applications are running; this is where the most active data is generated and consumed. As the high-performance tier fills, data that is no longer being actively used will be migrated to higher-capacity and generally lower-cost-per-terabyte storage platforms for long-term retention. Data migration is ideally managed automatically and transparently to the end user.

Intel® Enterprise Edition for Lustre provides a framework for incorporating a Lustre file system into an HSM implementation, as the high performance storage tier. When a new file is created, a replica can be made on the associated HSM archive tier, so that two copies of the file exist. As changes are made to the file, these are replicated onto the archive copy as well. The process of replicating data between Lustre and the archive is asynchronous, so there will be a delay in data generated on Lustre being reflected in the archive tier. As the available capacity is gradually consumed on the Lustre tier, the older, least frequently used files are "released" from Lustre, meaning that the local copy is deleted from Lustre and replaced with a stub file that points to the archive copy. Applications are not aware of the locality of a file: there is no distinction between a file on the archive and a file on the Lustre file system, from the perspective of directory listings or the stat() system call. Crucially, applications do not need to be re-written in order to work with data stored on an HSM system. If a system call is made to open a file that has been released, the HSM software automatically dispatches a request to retrieve the file from the archive and restore it to the Lustre file system. This may be noticeable in the form of a delay, but is otherwise transparent to the application.

Figure 2 provides an overview of the hardware architecture for a typical IEEL HSM file system. The metadata servers have an additional process called the *HSM Coordinator* that accepts, queues and dispatches HSM requests (Other documents may refer to this as the MDT Coordinator. The HSM Coordinator runs on the MDS/MDT). HSM requests are submitted from Lustre clients, either through the command line or through a special-purpose third-party application known as a Policy Engine. The Policy Engine software makes use of Lustre's HSM API to interact with the HSM Coordinator. The HSM platform also requires an interface between the Lustre file system tier and the archive tier. Servers called HSM Agents (also known as Copytool servers) provide this interface.

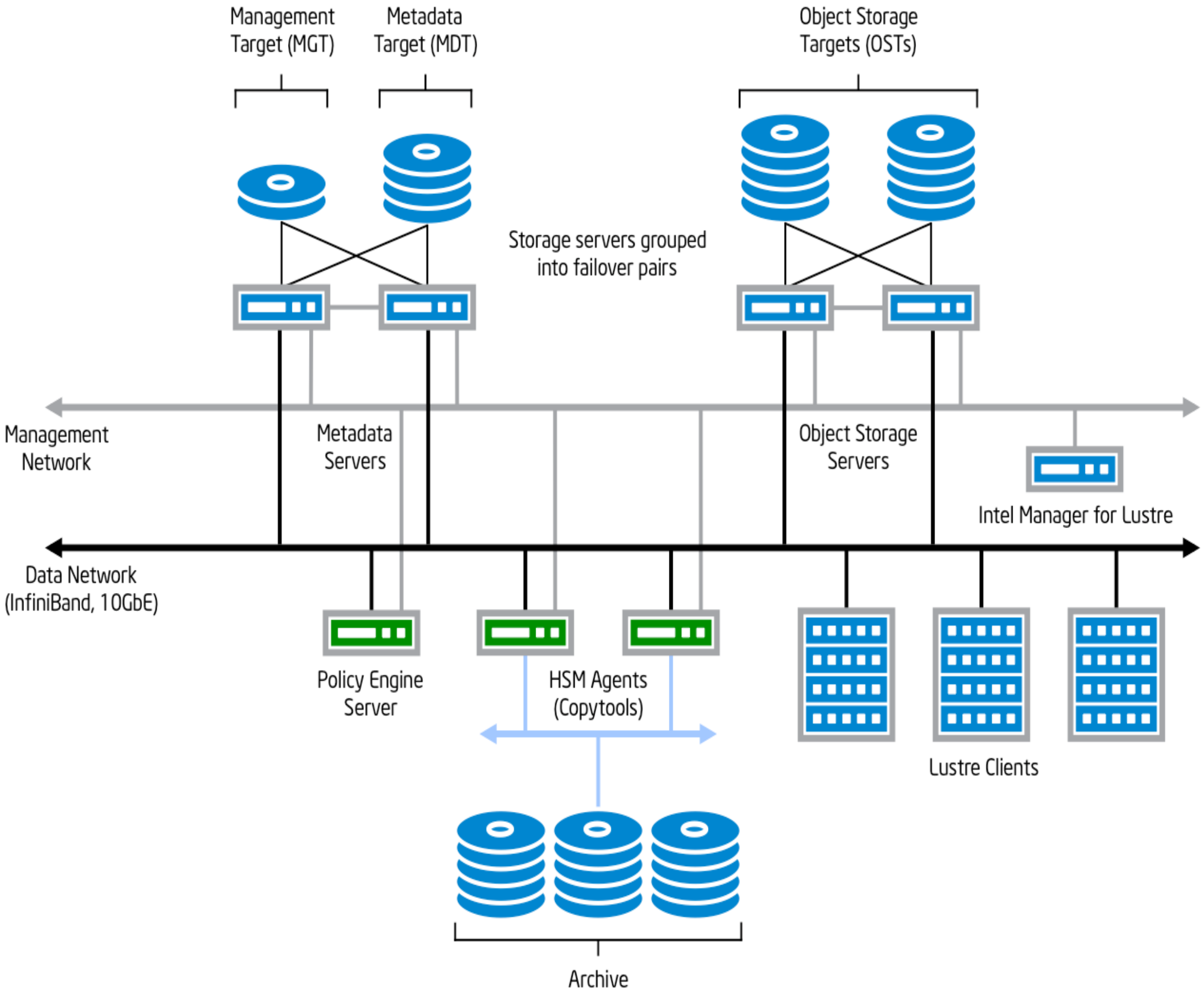


Figure 2. HSM Architecture

Intel® Enterprise Edition for Lustre* software provides reference implementations of both the Policy Engine and the HSM Copytool. The copytool supplied with Lustre uses POSIX interfaces to copy data to and from the archive. While this implementation of copytool is completely functional, it has been created principally as an example of how to develop copytools for other archive types, which may use different APIs. The copytool included with this software is intended as a reference implementation, and while it stable and reliable, it is not a high-performance tool and for that reason, may not be suitable for a “production environment”.

The Policy Engine distributed with Intel® Enterprise Edition for Lustre is called *Robinhood*. Robinhood is an open-source application with support for Lustre HSM. Robinhood tracks changes to the file system by registering with the Lustre MDT Changelogs service, and records this information persistently in a relational database. Robinhood then analyses this information and takes action based on a set of rules defined by the file system's maintainers.

These rules govern how files and directories will be managed by the HSM system, and the conditions under which to take an action. Among other things, policies in Robinhood determine how often to copy files to the archive tier, and the criteria for purging files from the Lustre tier in order to release capacity for new data.

Configuring HSM

There are three central tasks required to setup basic HSM for a Lustre file system created and managed by Intel® Manager for Lustre software:

1. Enable the HSM Coordinator process.
2. Provision an HSM Agent host.
3. Add a copytool instance to the HSM Agent host (and start copytool).

Robinhood setup and configuration is described later in this guide.

Enabling the HSM Coordinator

Before a Lustre file system can be integrated into a hierarchical storage management platform, the metadata server for the file system must be configured to run the HSM Coordinator process. This must happen before Copytool agents can be registered and HSM transactions can be processed.

Intel® Manager for Lustre software offers two ways to activate the HSM Coordinator for a Lustre file system:

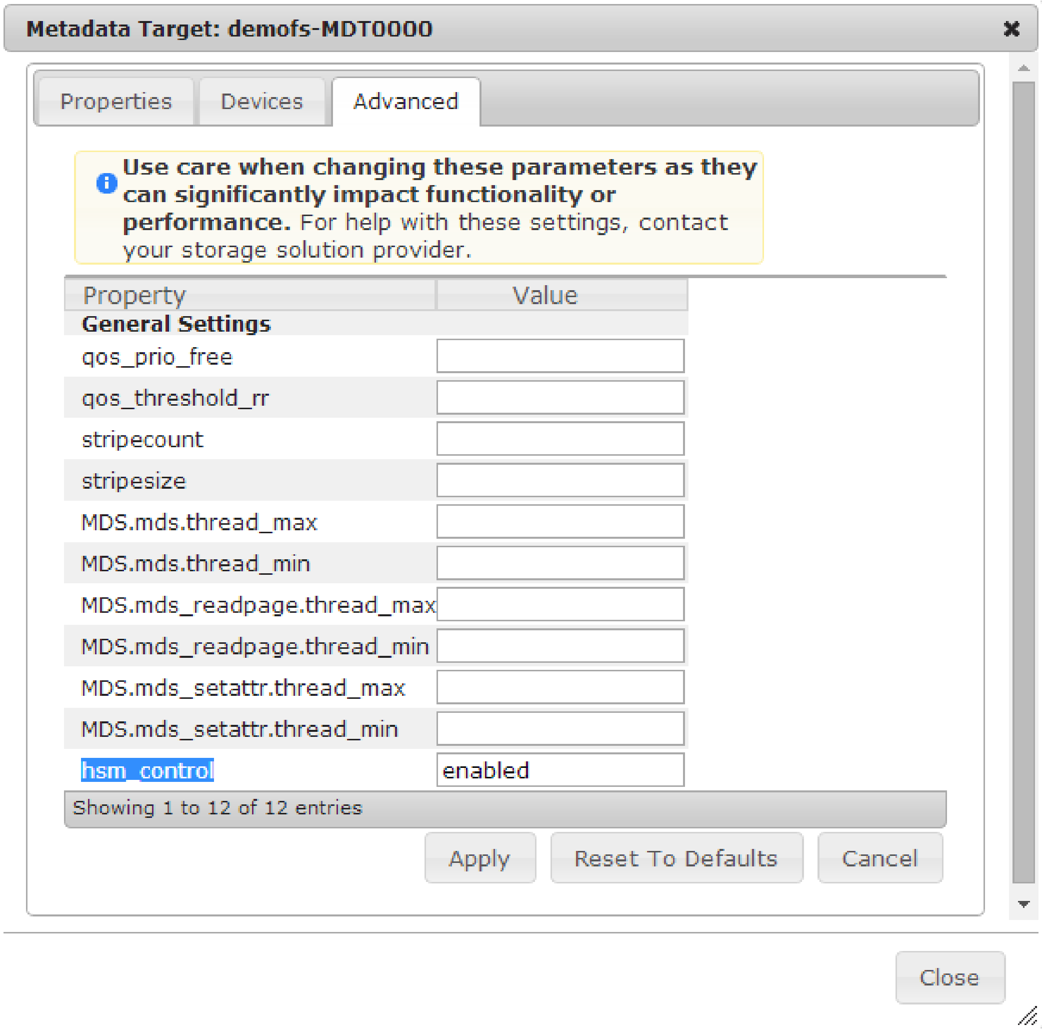
- for existing file systems, by editing the advanced parameters of the file system's Metadata Target (MDT), after the file system has been created
- when first creating the file system

Enabling the HSM Coordinator for an Existing File System

The HSM Coordinator can be enabled at any time after a file system has been created, provided that the file system is based on Lustre 2.5.0 or later, the Metadata service is online, and the MDT and MGT are mounted.

1. Log into Intel® Manager for Lustre dashboard as a superuser.
2. Navigate to the File Systems window: Click **Configuration** > **File Systems**.
3. Select the file system for which HSM support is to be enabled from the list of file systems.
4. Under Metadata Target, click the name of the metadata target.
5. Click the **Advanced** tab and look for the property **hsm_control**.

6. Enter the string "enabled" into the text field. For example:



Metadata Target: demofs-MDT0000

Use care when changing these parameters as they can significantly impact functionality or performance. For help with these settings, contact your storage solution provider.

Property	Value
General Settings	
qos_prio_free	
qos_threshold_rr	
stripecount	
stripesize	
MDS.mds.thread_max	
MDS.mds.thread_min	
MDS.mds_readpage.thread_max	
MDS.mds_readpage.thread_min	
MDS.mds_setattr.thread_max	
MDS.mds_setattr.thread_min	
hsm_control	enabled

Showing 1 to 12 of 12 entries

7. Click the **Apply** button. The background will briefly flash to grey then back to white to indicate that the setting has been applied. Click **Close** when done.

Enabling the HSM Coordinator when Creating the File System

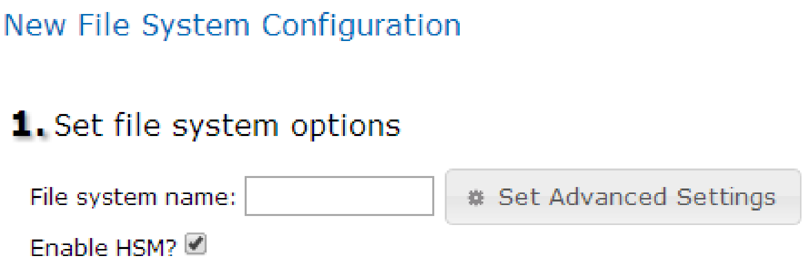
The HSM coordinator can be enabled when the Lustre file system is created using Intel® Manager for Lustre. See the online Help for the Intel® Manager for Lustre software for complete instructions on creating a Lustre file system. *Near the end of the process*, after adding servers, configuring HA failover, adding targets, etc., you will create the file system. At this point, the HSM Coordinator can be enabled.

(The following instructions are borrowed from the Intel Manager for Lustre online Help.)

To create the new Lustre file system:

1. At the menu bar, click the **Configuration** drop-down menu and click **File Systems** to display the File System Configuration page.

2. Click **Create File System** to display New File System Configuration.
3. In the File system name field, enter the name of the new file system. The name can be no more than eight characters long and should conform to standard Linux naming conventions.
4. If this file system is to utilize Hierarchical Storage Management, click the check-box **Enable HSM**.



New File System Configuration

1. Set file system options

File system name:

Enable HSM? ☑

Verifying the Current Status of the Coordinator

The most reliable way to verify the status of the HSM Coordinator process is to log into the metadata server that currently has the MDT mounted. Then enter the following command:

```
lctl get_param mdt.*.hsm_control
```

For example:

```
[root@ieel-mds1 ~]# lctl get_param mdt.*.hsm_control
mdt.demo-MDT0000.hsm_control=enabled
```

HSM Coordinator States

The `hsm_control` property has several options for managing the state of the process. These are:

- `enabled`: start the coordinator thread
- `disabled`: pause the coordinator. New requests will be queued but not scheduled
- `shutdown`: stop the coordinator. New requests cannot be submitted
- `purge`: clear all currently recorded requests. Do not change the coordinator state

Please refer to the Lustre Operations Manual, Chapter 22, for a comprehensive description of HSM features and the HSM Coordinator.

Provision an HSM Agent with the POSIX Copytool

HSM Agent hosts are special-purpose Lustre clients that have an interface to the HSM archive storage tier. This interface is managed by a software service known as the copytool that runs on each HSM Agent. Different archive storage platforms can have different interfaces or APIs, each of which will require a specific copytool. This document covers the configuration of an HSM Agent that will interface to a POSIX-compliant archive platform, using the POSIX copytool included with Intel® Enterprise Edition for Lustre.

Note: The copytool included with this software is intended as a reference implementation, and while it has performed acceptably at modest scale in our testing, it would be advisable to test thoroughly before using in a production environment

The archive storage solution can be built using any one of a number of technologies, whether based on a POSIX-compatible interface or another protocol. For scalability, multiple HSM agents can be established for a single archive, provided that each agent has equivalent access to the same archive tier.

In the examples that follow, the POSIX archive target is a locally-attached file system (this is not a scalable solution but is sufficient for the purposes of demonstration). For testing multiple agents connected to a single archive tier, a network-based POSIX target such as NFS or another Lustre file system could be used.

Preparation

HSM Agent hosts provide a bridge between the Intel® EE for Lustre file system and the archive storage tier. Prior to integrating the HSM Agent into an Intel® EE for Lustre file system, please ensure that the following prerequisites have been satisfied:

- The HSM Coordinator service has been enabled on the MDT for the file system (performed in chapter 2).
- The target HSM Agent host has been installed with a supported Linux-based operating system.
- The HSM Agent is able to participate as a Lustre client on a high performance data network. The HSM Agent must be able to communicate with the Lustre servers on this network.
- The HSM Agent host can be reached by the Intel® Manager for Lustre server over the management network and has been configured so that the manager can provision software either through a passphrase-less SSH connection or equivalent mechanism.
- The HSM Agent host has been configured with access to the archive storage platform. For a POSIX-compatible archive, ensure that the archive is mounted on the HSM Agent host and that the root superuser account has read- and write-privileges on that mount point. Ideally, the archive storage should be available automatically at system boot.

For very small-scale testing purposes, a locally-attached storage volume for a single HSM Agent is sufficient to act as the archive storage. One can create an EXT4 file system for this purpose and create an entry in `/etc/fstab` so that it mounts on boot. For example, assuming that a storage volume containing a single partition exists on `/dev/sdb`:

```
mkdir -p /archive/demo
mkfs.ext4 -L ARC1 /dev/sdb1
printf "LABEL=ARC1\t\t/archive/demo\text4\tdefaults\t0 0\n" \
  >>/etc/fstab
```

Or equivalently:

```
echo "LABEL=ARC1        /archive/demo     ext4 default   0 0"
>>/etc/fstab
```

Then mount the file system as normal:

```
mkdir -p /archive/demo
mount -L ARC1
```

In the above example, the archive file system was given a label to help identify the storage volume.

The above trivial example is suitable only for development and testing purposes, but it is sufficient to provide system integrators and maintainers with exposure to the tools and processes of Lustre's HSM implementation.

A production HSM archive may be far larger than the Lustre file system itself and will need to be serviced by multiple HSM Agent nodes in order to be able to meet scalable performance demands. This means that the archive platform should be accessible simultaneously from multiple HSM agent servers. Implementation of a production-quality archive is beyond the scope of this document.

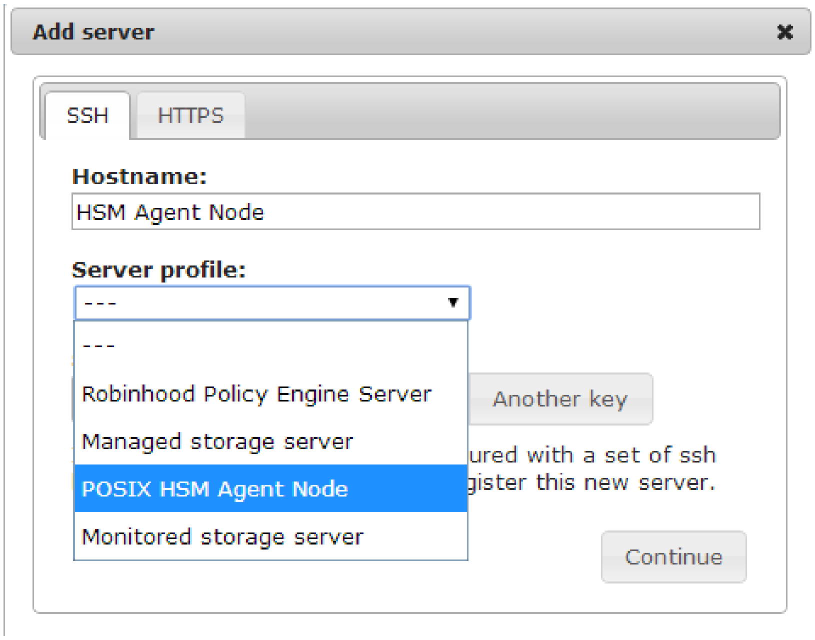
Add an HSM Agent Host with Intel® Manager for Lustre* Software

Intel® Enterprise Edition for Lustre 2.n uses the concept of work nodes. These are special-purpose systems that are not core Lustre file system servers (in other words, these are systems that are not Metadata or Object Storage servers; usually, worker nodes are special-purpose Lustre clients). HSM Agents are one of the worker node types that can be configured and managed from the Intel® Manager for Lustre GUI.

To add an HSM Agent:

1. Log into the Intel® Manager for Lustre dashboard as a superuser.
2. Navigate to the Servers window: Click **Configuration** > **Servers**.

3. Click **+Add Server**.
    a. Enter the name of the host to be used as the HSM Agent.
    b. Under Server profile, select POSIX HSM Agent Node.



Hostname: HSM Agent Node

Server profile: ---

ured with a set of ssh
gister this new server.

    c. Select the appropriate authentication scheme and click **Continue**.

4. When the target host passes the prerequisites check, click **Confirm** to provision the HSM Agent. If any of the checks fail, please rectify before continuing. Common issues include incorrect permissions (for example, a missing SSH key), incomplete host name resolution and incorrect or incomplete YUM repository configuration.
5. The Add Server process will install the Lustre client software and kernel modules, and automatically resolve all package dependencies during the installation process. The software installation process relies on YUM and access to operating system repositories, including update repositories, in order to successfully complete installation. Note that the Linux kernel for the target host may also be updated by the installation process if the currently-installed kernel does not match the version required by the Lustre kernel modules.

Next, configure the LNET settings for this HSM Agent node:

1. At the menu bar, click the **Configuration** drop-down menu and click **Servers**.
2. Identify the HSM Agent node.
3. To set the NID for a network interface on a given server, the **LNet State** for that server must indicate **LNet up**. To load and start LNet, click **Actions** and select **Start LNet**.

4. When LNet has started, the **LNet State** indicates **LNet up**, and the **Configure** button becomes active. Click **Configure**.
5. The NID Configuration for <server name> dialogue appears. This window applies to this server only. Available network interfaces appear on the left, with their associated IP addresses. Click the LNet button for the desired interface to select the Lustre Network number you want to assign to this interface. Do this for each interface you want to configure with an LNet NID.
6. Click **Save**. Click **Close** to close this dialogue.

Verify the settings by logging into the target host and viewing the contents of the file, `/etc/modprobe.d/iml_lnet_module_parameters.conf`, and by running the command `lctl list_nids`. For example:

```
[root@ieel-c004 ~]# cat
/etc/modprobe.d/iml_lnet_module_parameters.conf
# This file is auto-generated for Lustre NID configuration by IML
# Do not overwrite this file or edit its contents directly
options lnet networks=tcp0(eth1)


### LNET Configuration Data
### {
###   "state": "lnet_unloaded",
###   "modprobe_entries": [
###     "tcp0(eth1)"
###   ],
###   "network_interfaces": [
###     [
###       "10.20.73.34",
###       "tcp",
###       0
###     ]
###   ]
### }
[root@ieel-c004 ~]# lctl list_nids
10.20.73.34@tcp
```

Configure the HSM Copytool Daemon

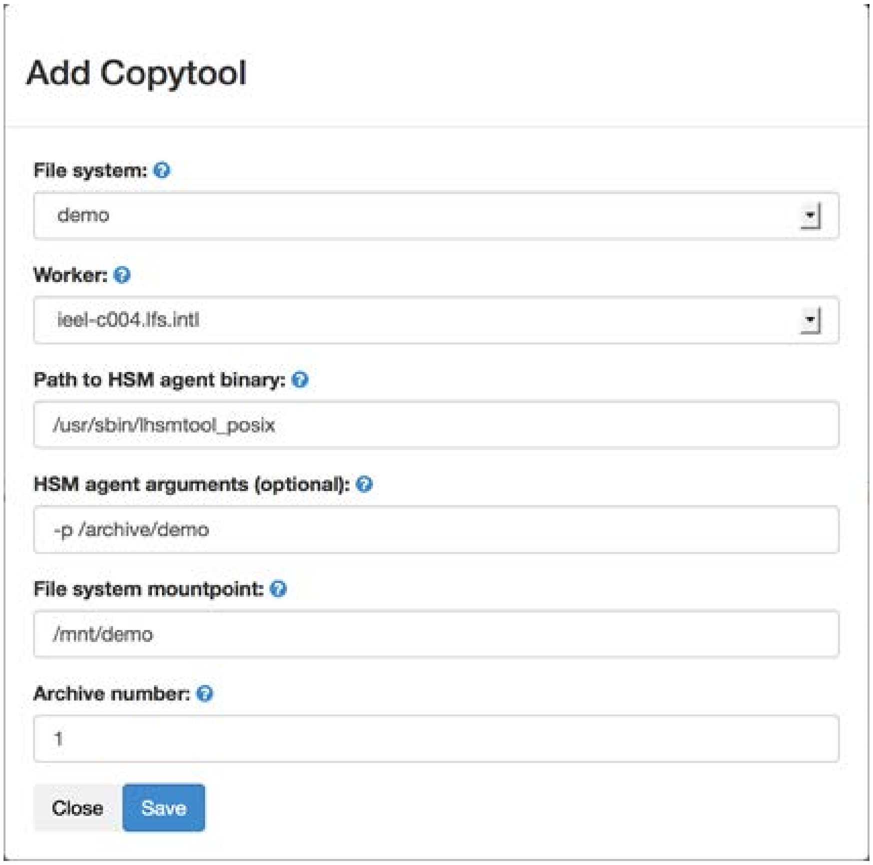
The Intel® Manager for Lustre* software can configure and manage the HSM copytool processes that run on Agent hosts, and this is the preferred method for deploying the copytools within an Intel® EE for Lustre environment. This document discusses the POSIX copytool provided as a reference implementation with Intel® EE for Lustre. Other copytool applications may have different requirements and configurable options. Refer to the copytool supplier for instructions specific to third-party implementations.

To configure the POSIX Copytool on an HSM Agent host managed by Intel® Manger for Lustre:

1. Log into the Intel® Manager for Lustre dashboard using a superuser account.
2. Navigate to the HSM management window: Click **Configuration** >**HSM**.
3. At the bottom of the HSM window, click **+Add Copytool**.
4. At the Add Copytool form, enter the following:
    a. File system: Specify file system for which this copytool will perform HSM actions.
    b. Worker: This is the POSIX HSM Agent node that was configured in Add an HSM Agent node. Each copytool instance has its own Agent node, so there may be several.
    c. Path to the HSM agent binary: Enter the full command line path to the copytool software. The POSIX copytool distributed in the Intel® EE for Lustre packages is installed at the following path: `/usr/sbin/lhsmtool_posix`. If an alternative copytool binary is to be used, or the copytool has been moved to a different location, enter the appropriate path here.
    d. HSM agents arguments (optional): While this field may be labeled as *optional*: this is *not strictly true*. Some HSM copytools may not require additional inputs, but the POSIX copytool *must be told* where the archive storage tier is located (the archive mount point). To do this for the POSIX copytool, use the -p <path> flag.

       Note: Do not provide any flags that will cause the copytool process to be run in the background (e.g. --daemon); this interferes with the ability of Intel® Manager for Lustre software to control and monitor the copytool process.
    e. File system mount point: The Lustre file system mount point on the worker node. Copytool instances require client access to the associated Lustre file system.
    f. Set the archive number appropriately: if there is only one archive available for the file system, set the archive number to "1" (the default).
    g. Click **Save** when ready to commit the configuration.

The following is a completed example for configuring the POSIX copytool on a file system called "demo" with an HSM Agent "ieel-c004.lfs.intl" that was previously added to the Intel® Manager for Lustre server configuration as an HSM Agent.



Add Copytool

File system: demo

Worker: ieel-c004.lfs.intl

Path to HSM agent binary: /usr/sbin/lhsmtool_posix

HSM agent arguments (optional): -p /archive/demo

File system mountpoint: /mnt/demo

Archive number: 1

Start the Copytool

When a copytool is added to an Intel® EE for Lustre file system configuration, it is not automatically activated. Instead, the copytool will initially be set to the state *Unconfigured*. The configuration exists inside the Intel® Manager for Lustre database, but it has not been applied directly to the target HSM Agent.

At the bottom of the HSM configuration page, click the **Actions** drop down menu, next to the required copytool instance, and select **Start**. The copytool status will change from "Unconfigured" to "Idle" and the graph will register that a new idle copytool instance has been added and is running on the file system.

Check that the Copytool Process is Running

To verify that the copytool is actually running on the HSM Agent, log into the agent host and run the following command:

```
ps -ef | grep hsm
```

If the copytool is running, the command will return output equivalent to the following:

```
[root@ieel-c004 /]# ps -ef | grep hsm
root      7126     1  0 01:40 ?        00:00:00 /usr/sbin/lhsmtool_posix --quiet --
update-interval 5 --event-fifo /var/spool/lhsmtool_posix-demo-1-0-events --archive 1 -
p /archive/demo /mnt/demo
root      7169  6298  0 01:43 pts/0    00:00:00 grep hsm
```

Notice that the Intel® Manager for Lustre software has supplied a number of arguments to the copytool application, in addition to the -p <path> for the storage archive and the archive identifier. Please avoid trying to overload these parameters with their own options, as it may have a negative impact on the stability of the copytool application.

Check that the Copytool Instance is Registered with the Coordinator

To verify that the copytool instance is registered with the HSM Coordinator for the file system, log into the metadata server that has the MDT mounted as root and issue the following command:

```
lctl get_param mdt.*.hsm.agents
```

This command will return a line of output for each copytool that is currently registered. For example:

```
[root@ieel-mds1 ~]# lctl get_param mdt.*.hsm.agents
mdt.demo-MDT0000.hsm.agents=
uuid=3f9e4c18-46a7-cf21-7a6f-0d899b9a6ead archive_id=1
requests=[current:0 ok:0 errors:0]
```

Running the POSIX Copytool Outside of Intel® Manager for Lustre* Software

It is possible to deploy HSM copytool processes to agent hosts without using the Intel® Manager for Lustre* software. However, this is strongly discouraged and may lead to inconsistencies in the deployment. Copytools that are not installed by Intel® EE Lustre* software cannot be maintained by the Intel® Manager for Lustre* interface and it is not possible to monitor copytool status, availability or performance.

Quick Verification Check and Introduction to the HSM Command Line

When the copytool starts, the HSM coordinator will send HSM instructions for archiving and restoring files from the archive through the copytools. If there are no copytools running on the Lustre file system, the HSM coordinator will queue transaction requests until at least one copytool for the archive comes online.

The following process can be used to verify the system is working correctly and to demonstrate some of the features of Lustre HSM with the POSIX copytool:

1. Login to a Lustre client computer (but not an HSM Agent) and create a new file on the Lustre file system. For example:

   ```
   [root@c64-1 ~]# df -ht lustre

   Filesystem            Size  Used Avail Use% Mounted on
   10.70.73.12@tcp0:10.70.73.11@tcp0:/demo

                        20G  876M   18G   5% /mnt/demo

   [root@c64-1 ~]# cd /mnt/demo

   [root@c64-1 demo]# dd if=/dev/zero of=f001 bs=1M count=1

   1+0 records in

   1+0 records out

   1048576 bytes (1.0 MB) copied, 0.00550202 s, 191 MB/s ?
   ```

2. Examine the HSM state of the newly created file:

   ```
   [root@c64-1 demo]# lfs hsm_state f001

   f001: (0x00000000)
   ```

   This is the normal condition for a newly created file that has not been registered with the HSM archive. None of the flags have been set and the file has no reported state with respect to HSM.

3. Archive the file:

   ```
   [root@c64-1 demo]# lfs hsm_archive --archive 1 f001
   ```

4. Examine the file again on the Lustre client:

   ```
   [root@c64-1 demo]# lfs hsm_state f001

   f001: (0x00000009) exists archived, archive_id:1
   ```

5. Log into the HSM Agent running the POSIX copytool and look at the contents of the archive file system:

```
[root@c64-2a ~]# find /archive/demo/

/archive/demo/

/archive/demo/0001
```

```
/archive/demo/0001/0000
/archive/demo/0001/0000/0400
/archive/demo/0001/0000/0400/0000
/archive/demo/0001/0000/0400/0000/0002
/archive/demo/0001/0000/0400/0000/0002/0000
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0.lov
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
/archive/demo/shadow
/archive/demo/shadow/f001
```

With the POSIX copytool implementation, the FID of the file is mapped into the archive's directory structure and the content copied into the file: `0x200000400:0x1:0x0`. Lustre metadata information (including stripe information) is captured into the file: `0x200000400:0x1:0x0.lov`. There is also a `shadow` directory. This provides a human-readable view of the archive and replicates the file system structure (the name space) of the Lustre file system as seen on the Lustre clients. Each file in the shadow tree is a soft link to the FID reference in the archive. For example:

```
[root@c64-2a ~]# ls -l /archive/demo/shadow/f001
lrwxrwxrwx 1 root root 52 Aug  3 04:21 /archive/demo/shadow/f001
-> ../0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
```

6. Log into the normal Lustre client and change the content of the file. Observe the change in state:

   ```
   [root@c64-1 ~]# cd /mnt/demo
   [root@c64-1 demo]# dd if=/dev/zero of=f001 bs=1M count=2
   2+0 records in
   2+0 records out
   2097152 bytes (2.1 MB) copied, 0.010045 s, 209 MB/s
   [root@c64-1 demo]# lfs hsm_state f001
   f001: (0x0000000b) exists dirty archived, archive_id:1
   ```

   Notice that the archived copy is now marked as dirty.

7. Update the archive copy:

   ```
   [root@c64-1 demo]# lfs hsm_archive --archive 1 f001
   [root@c64-1 demo]# lfs hsm_state f001
   f001: (0x00000009) exists archived, archive_id:1
   ```

Now the dirty flag has been cleared.

8. Remove the file from the archive:

```
[root@c64-1 demo]# lfs hsm_remove f001
[root@c64-1 demo]# lfs hsm_state f001
f001: (0x00000000), archive_id:1
[root@c64-1 demo]# ls
f001
```

The file has not been deleted from Lustre, but it is no longer archived. Note that the archive identifier is still set on the file, even though it is no longer archived.

9. Create a new file, much larger than the last one and archive it:

```
[root@c64-1 demo]# dd if=/dev/zero of=f002 bs=1M count=2048
[root@c64-1 demo]# lfs hsm_archive --archive 1 f002
```

Notice that the archive command returns almost immediately.

10. It will take time to archive the file. Watch the progress through the IML manager or directly on the HSM agent host. The `lfs hsm_state` command can also be used to monitor the file from a Lustre client:

```
[root@c64-1 demo]# lfs hsm_state f002
f002: (0x00000001) exists, archive_id:1
...
[root@c64-1 demo]# lfs hsm_state f002
f002: (0x00000009) exists archived, archive_id:1
```

The copy to the archive is not complete until `hsm_state` lists both `exists` and `archived`.

11. Run the `lfs hsm_release` command for the new file. Observe the reported file system size before and after the command is run:

```
[root@c64-1 demo]# df -ht lustre
Filesystem            Size  Used Avail Use% Mounted on
10.70.73.12@tcp0:10.70.73.11@tcp0:/demo
                      9.9G  2.5G  7.0G  27% /mnt/demo
[root@c64-1 demo]# lfs hsm_release f002
[root@c64-1 demo]# lfs hsm_state f002
f002: (0x0000000d) released exists archived, archive_id:1
[root@c64-1 demo]# df -ht lustre
```

```
Filesystem            Size  Used Avail Use% Mounted on
10.70.73.12@tcp0:10.70.73.11@tcp0:/demo
                      9.9G  440M  9.0G   5% /mnt/demo
```

The released file has been removed from the Lustre file system and replaced with a stub file referencing the archived copy. Capacity has been freed as a result. `lfs hsm_release` is asynchronous, so this may not happen immediately.

12. Even though the file has been released, the perceived size of the file remains unchanged:

```
[root@c64-1 demo]# ls -lh f002
-rw-r--r-- 1 root root 2.0G Aug 10 09:34 f002
```

To client applications, f002 still looks like a 2GiB file.

13. Restore the archived file to the Lustre file system. This can be done using the `lfs hsm_restore` command, or by simply trying to open or access the file:

```
[root@c64-1 demo]# lfs hsm_restore f002
[root@c64-1 demo]# df -ht lustre
Filesystem            Size  Used Avail Use% Mounted on
10.70.73.12@tcp0:10.70.73.11@tcp0:/demo
                      9.9G  2.5G  6.9G  27% /mnt/demo
[root@c64-1 demo]# lfs hsm_state f002
f002: (0x00000009) exists archived, archive_id:1
```

The `lfs hsm_restore` command is asynchronous, but attempting to access a released file will block until the file is fully restored. For example:

```
[root@c64-1 demo]# ls -lh f002
-rw-r--r-- 1 root root 2.0G Aug 10 09:34 f002
[root@c64-1 demo]# lfs hsm_release f002
[root@c64-1 demo]# lfs hsm_state f002
f002: (0x0000000d) released exists archived, archive_id:1
[root@c64-1 demo]# df -ht lustre
Filesystem            Size  Used Avail Use% Mounted on
10.70.73.12@tcp0:10.70.73.11@tcp0:/demo
                      9.9G  440M  8.9G   5% /mnt/demo
[root@c64-1 demo]# time file f002
f002: data
```

```
real    0m31.078s
user    0m0.025s
sys 0m0.006s
[root@c64-1 demo]# time file f002
f002: data

real    0m0.008s
user    0m0.003s
sys 0m0.003s
```

In this example, running the `file` command on the released file takes nearly 33 seconds the first time it is run because the command is blocked until the HSM restore operation completes. Running the `file` command a second time returns almost immediately because the file has already been restored.

Working with HSM Archive Identifiers

The Default HSM Archive

The archive identifier is used to indicate that a copytool instance is bound to a specific archive. If no archive is specified on the copytool command line, the default archive identifier is read from the MDT for the Lustre file system. The Lustre file system client commands (`lfs hsm_*`) also need to make reference to an archive identifier in order to inform the HSM Coordinator of the intended target archive storage tier to use. As with the copytool, HSM client commands will use the file system default archive identifier if one is not explicitly referenced.

To determine the default identifier, log into the MDS and run the following command:

```
lctl get_param mdt.*.hsm.default_archive_id
```

This will return the value of the default archive identifier for all the MDTs on the host. The asterisk in the command line is a wild card character and can be replaced with the label of a specific MDT instead. The default value for this parameter is 1. For example, on a newly created file system, called demo:

```
[root@m64-1 ~]# lctl get_param mdt.*.hsm.default_archive_id
mdt.demo-MDT0000.hsm.default_archive_id=1
```

To change the value of the default archive identifier, use the `lctl set_param` command:

```
lctl set_param [-P] mdt.<fsname>-
MDT<index>.hsm.default_archive_id=<archive id>
```

For example, to set the default archive identifier to "2", on a Lustre file system called "demo":

```
lctl set_param -P mdt.demo-MDT*.hsm.default_archive_id=2
```

Please note that this parameter is volatile and will be lost when the MDT is unmounted, unless the `-P` flag (for "persistent") is specified. Also note that in order to set the parameter persistently, the command must be executed on the host that is running the MGS for the file system, not the MDS.

There is a special archive identifier, called archive 0 that was originally used as a default for Lustre HSM transactions and to provide a catch-all archive. This is explained more fully in the next section, but in general, *it is not recommended to rely on or make use of archive identifier 0*.

On production systems, all archives should be explicitly allocated a positive integer identifier (e.g. through the `--archive` flag for the POSIX copytool startup command) and, all `lfs hsm_*` commands should reference this ID number when executing HSM instructions. This approach makes the configuration easier to design, document and understand, particularly when there are other applications that need to be configured, such as the Policy Engine, and when there are multiple HSM archives registered with the Lustre file system.

If a request is sent to an archive ID that is not registered with the HSM coordinator, the request will hang indefinitely, or until such time as a copytool instance for the archive becomes available. So, if a user sends a request to a non-existent archive, the request will never complete and the file will not be copied to the archive. This is unless there is a copytool for archive 0, in which case the user can specify any archive ID they like and the data will always be sent to the copytool[s] for archive 0. This is potentially dangerous behavior, as all of the files may appear to be stored in different archives (according to Lustre) but will in fact be held within a single data store. It is recommended that the `default_archive_id` is never set equal to zero.

Archive Zero – The "Any" Archive

The zero archive has a special meaning in HSM: If a copytool running on an HSM Agent is registered with the HSM coordinator thread using an archive identifier of 0, it means that the copytool will accept HSM actions on behalf of any archive. Prior to the release of Lustre version 2.5.0, archive 0 was also the default if an archive was not specified by the HSM command line tools; this was changed for the 2.5 release of Lustre in order to make the behavior of the HSM tools more consistent.

Strictly speaking, there is no such thing as archive zero – it is effectively the "any" archive. To illustrate this point, consider the following example.

Caution: The following example is for illustrative purposes only and may compromise the HSM service if attempted in a live environment. **Do not** run this demonstration on a production file system.

1. Log into Intel® Manager for Lustre using a superuser account and shut down all of the running copytool processes.
2. Log into the MDS server and make sure that the MDT has no copytool agents registered:

   ```
   lctl get_param mdt.*.hsm.agents
   ```

   The list should be empty.
3. Return to the Intel® Manager for Lustre configuration and create a new POSIX copytool instance with an archive identifier of zero. Start the copytool.
4. Back on the MDS server, the HSM Coordinator will now have an entry for the copytool agent in procfs, e.g.:

   ```
   [root@m64-1 ~]# lctl get_param mdt.*.hsm.agents

   uuid=c6bd632a-a92a-f13c-f15a-1674133497ac archive#=0 (all) r=0
   s=0 f=0
   ```

Note that the third column has the word "all" parenthesized. This tells the coordinator to send all HSM commands to this archive, regardless of the archive identifier referred to by the client in its request.

5. Log into a standard Lustre client and create a new file on the Lustre file system, e.g.:

   ```
   [root@c64-1 ~]# df -ht lustre

   Filesystem            Size  Used Avail Use% Mounted on

   10.70.73.12@tcp0:10.70.73.11@tcp0:/demo

                         9.9G  438M  9.0G   5% /mnt/demo

   [root@c64-1 ~]# cd /mnt/demo

   [root@c64-1 demo]# dd if=/dev/zero of=f001 bs=1M count=1

   1+0 records in

   1+0 records out

   1048576 bytes (1.0 MB) copied, 0.00676696 s, 155 MB/s
   ```

6. Use the HSM command line interface to archive the file, making sure to use the --archive flag to select an archive identifier other than 0 (zero):

   ```
   [root@c64-1 demo]# lfs hsm_state f001
   ```

```
f001: (0x00000000)
[root@c64-1 demo]# lfs hsm_archive --archive 1 f001
[root@c64-1 demo]# lfs hsm_state f001
f001: (0x00000009) exists archived, archive_id:1
```

7. Check the contents of the archive storage on the HSM Agent:

```
ls -l /archive/demo/shadow/f001
[root@c64-3a /]# find /archive
/archive
/archive/demo
/archive/demo/shadow
/archive/demo/shadow/f001
/archive/demo/lost+found
/archive/demo/0001
/archive/demo/0001/0000
/archive/demo/0001/0000/0400
/archive/demo/0001/0000/0400/0000
/archive/demo/0001/0000/0400/0000/0002
/archive/demo/0001/0000/0400/0000/0002/0000
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0.l
ov
[root@c64-3a /]# ls -l /archive/demo/shadow/f001
lrwxrwxrwx 1 root root 52 Aug  1 19:40 /archive/demo/shadow/f001
-> ../0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
[root@c64-3a /]# ls -lL /archive/demo/shadow/f001
-rw-r--r-- 1 root root 1048576 Jul 31 23:02
/archive/demo/shadow/f001
[root@c64-3a /]# md5sum
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
b6d81b360a5672d80c27430f39153e2c
/archive/demo/0001/0000/0400/0000/0002/0000/0x200000400:0x1:0x0
```

The `md5sum` ought to match the original file as well.

8. To clean up, log into the Intel® Manager for Lustre dashboard. Shut down and remove the HSM copytool instance that was created using archive identifier 0 (zero). Restore any previous copytools to their original run state.

From this it can be observed that the HSM Agent registered using an archive identifier of 0 processed a request that was specifically targeted at an archive with identifier 1. It would also have processed requests for archive identifier 2, 3 or indeed, any archive identifier. While this may have some useful applications, it can also cause confusion and may lead to unanticipated results.

The order in which copytools and the archives are registered with the coordinator is also significant. If an archive with ID 0 is registered with the HSM coordinator by a copytool before any other archive, then all commands will always use archive 0, even if a different ID is requested and that ID is listed in `/proc/fs/lustre/mdt/<fsname>-<MDT>/hsm/agents` on the MDS.

Additional Notes

It is possible to run multiple copytools on the same HSM agent node; the software does not prevent this behavior. However, running multiple copytools from the same HSM agent node is not supported and is discouraged. This may be useful when testing the HSM mechanism and there may be some environments where running multiple copytools per host delivers a benefit. However, the general guidance is that copytools should be limited to one instance per HSM agent.

The HSM command line tools only operate on files; any attempt to mark a directory for archiving will fail. Newly-created files are not automatically archived and must be added explicitly. For automated and comprehensive management of a Lustre HSM system, employ a Policy Engine application such as Robinhood.

Robinhood Policy Engine Installation

A policy engine is a software application acting on behalf of a human operator in support of storage management. The policy engine:

- collects data about the state of the Lustre file system by reading the MDT changelogs
- executes tasks based on rules applied to the collected data. The rules are referred to as policies.

The Robinhood policy engine is a software application used to manage Lustre file systems that participate in a Hierarchical Storage Management (HSM) tiered storage platform. Robinhood provides administrators with tools to describe storage policies to actively manage the content of Lustre file systems. Policies are used to make decisions about archiving files and managing available storage capacity. Robinhood also provides monitoring functionality and keeps records of Lustre's changelogs.

This chapter guides readers through the steps necessary to install and setup a Robinhood server on a file system running Intel® Enterprise Edition for Lustre software

Preparation

Robinhood runs on a dedicated system that is participating on a computer network as a Lustre client node. Prior to installing and configuring Robinhood, some preparation is required. Please ensure that the following prerequisites have been satisfied:

- The operating system for the machine has been installed in accordance with general best practice guidelines for the vendor's operating system distribution and the requirements in *Intel® Enterprise Edition for Lustre Software – Partner Installation Guide*. In that guide, also see the *section Creating a Managed Lustre File System – Configuring Clients*.
- Ensure that the Robinhood policy engine host can be reached by the Intel® Manager for Lustre server over the management network and has been configured so that the manager can provision software either through a passphrase-less SSH connection or other supported mechanism.
- An Intel® EE for Lustre file system has been created.
- The Policy Engine host is able to participate as a Lustre client on a high performance data network. The Policy Engine host must be able to communicate with the Lustre servers on this network.
- The HSM Coordinator has been enabled on the file system's MDS.
- At least one IEEL HSM Agent has been provisioned (e.g. POSIX copytool) and is running.

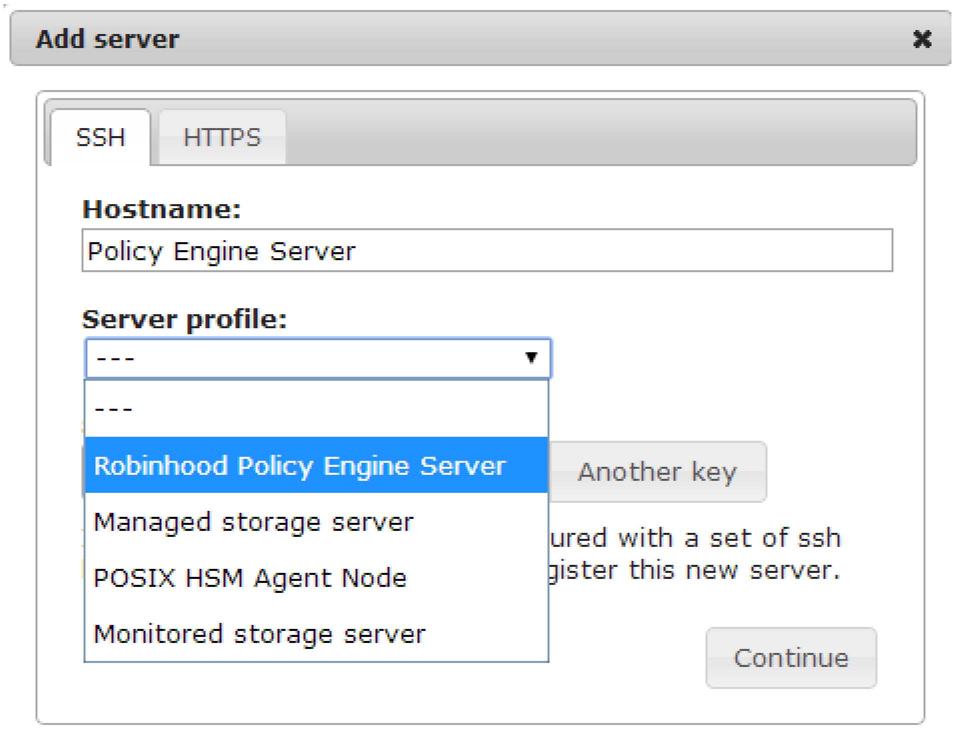
Provision the Robinhood Packages Using Intel® Manager for Lustre.

Intel® Enterprise Edition for Lustre 2.n introduced the concept of work nodes. These are special purpose systems that provide services to clients but which are not core Lustre servers. In other words, these are systems that are not Metadata or Object Storage servers. Robinhood is one of the worker node types that are available in Intel® EE for Lustre and can be provisioned from the Intel® Manager for Lustre application. Intel® Manager for Lustre does not provide configuration of Robinhood, but does provide a clean mechanism for installing Robinhood along with its dependencies.

To install the Robinhood Policy Engine on a host:

1. Log into the Intel® Manager for Lustre dashboard as a superuser.

2. Navigate to the Servers management window (**Configuration** > **Servers**).
3. Click **+Add Server**, and fill in the form as follows:
   a. Type in the name of the host to be used as the Robinhood server
   b. Select the Robinhood Policy Engine server profile:



Hostname: Policy Engine Server

Server profile: ---

ured with a set of ssh
gister this new server.

   c. Select the appropriate authentication scheme and click **Continue**.
4. If the target host passes the prerequisites check, click **Confirm** to provision Robinhood. If any of the checks fail, please rectify before continuing. Common issues include incorrect permissions (for example, a missing SSH key), incomplete host name resolution, and incorrect YUM repository configuration.
5. In addition to installing Robinhood and the MySQL relational database Robinhood uses to record data, the **Add Server** process will also install the Lustre client software and kernel modules, and will automatically resolve all package dependencies during the installation process. Software installation relies on YUM, and requires access to operating system repositories, including update repositories, to be able to successfully complete installation. Note that the Linux kernel for the target host may also be updated by the installation process if the currently installed kernel does not match the version required by the Lustre kernel modules.

Next, you need to configure LNET settings for this Robinhood policy engine server. Perform these steps:

1. At the menu bar, click the **Configuration** drop-down menu and click **Servers**.

2. Identify the Robinhood policy engine server.
3. To set the NID for a network interface on a given server, the **LNet State** for that server must indicate **LNet up**. To load and start LNet, click **Actions** and select **Start LNet**.
4. When LNet has started, the **LNet State** indicates **LNet up**, and the **Configure** button becomes active. Click **Configure**.
5. The NID Configuration for <server name> dialogue appears. This window applies to this server only. Available network interfaces appear on the left, with their associated IP addresses. Click the LNet button for the desired interface to select the Lustre Network number you want to assign to this interface. Do this for each interface you want to configure with an LNet NID.
6. Click **Save**. Click **Close** to close this dialogue.

You can verify the settings by logging into the target host and viewing the contents of the file, `etc/modprobe.d/iml_lnet_module_parameters.conf`, and by running the `lctl list_nids` command. For example:

```
[root@ieel-pe ~]# cat
/etc/modprobe.d/iml_lnet_module_parameters.conf

# This file is auto-generated for Lustre NID configuration by IML

# Do not overwrite this file or edit its contents directly

options lnet networks=tcp0(eth1)

... [additional output omitted]

[root@ieel-pe ~]# lctl list_nids

10.20.73.34@tcp
```

Mount the Lustre file system on the Robinhood Server

The Lustre file system must be mounted on the Robinhood server. To obtain the command to mount the file system, perform these steps:

1. At the manager Dashboard menu bar, click the **Configuration** drop-down menu and click **File Systems**.
2. Each Lustre file system created using Intel® Manager for Lustre is listed. Select the file system to be mounted. A page opens showing information for that file system.
3. On the file system page, click **View Client Mount Information**. The mount command to be used to mount the file system is displayed. Following is an example only:

   ```
   mount -t lustre 10.214.13.245@tcp0:/test /mnt/demo
   ```

4. On the Robinhood server, create the mount point for the file system (this is the last field in the mount command above). For example:

```
mkdir -p /mnt/demo
```

5. On the Robinhood server, enter the actual mount command from step 3, substituting your mount point.

Configure MDS Changelogs

Robinhood relies on the MDT Changelogs feature in Lustre in order to track changes to the file system. The Changelog records changes to the file system's metadata, such as creating and deleting files, modifying a file's content, ownership, permissions, and other attributes. Robinhood captures this data and stores it in a MySQL database for processing by its policy engine. Applications that want to consume the Changelogs content need to register with the MDT, which will return a unique identifier, referred to as a *userid*. This is not the same as a UNIX user account identifier; it is internal to Lustre and is a token used to distinguish registrations from multiple consumers. Each registered userid has its own view of the Changelog register. Changelog entries are kept until each registered user has acknowledged that the entry has been consumed.

The following steps will configure Changelogs for Robinhood:

1. Set the Changelog event mask on the MDS, ensuring that HSM-related events are registered. This must be executed on the MDS server. Avoid setting the event mask to "all", as some of the events (e.g. ATIME) can generate a large amount of changelog activity, impacting performance. One may wish to re-enable the XATTR setting if policies will be created based on this attribute. The following mask has been recommended by the HSM developers:

   ```
   lctl set_param [-P] \
     mdd.<fsname>-MDT*.changelog_mask="all-XATTR-MARK-ATIME"
   ```

   This mask requests that all events except XATTR, MARK and ATIME are monitored by the MDT Changelogs service. Use the `-P` flag to make sure that the changes are recorded persistently in the MGT. Here is a complete example:

   ```
   [root@m64-1 ~]# lctl set_param -P \
     mdd.demo-MDT*.changelog_mask="all-XATTR-MARK-ATIME"
   mdd.demo-MDT0000.changelog_mask=all-XATTR-MARK-ATIME
   ```

   Note: the `lctl set_param -P` command must be executed on the server that currently has the MGT mounted, otherwise the command will fail and settings will not be saved persistently. For temporary changes, run the `lctl set_param` command directly on the metadata target being changed.

2. Verify that the changelog mask has been set (this command must be run on the MDS):

```
[root@m64-1 ~]# lctl get_param mdd.demo-MDT*.changelog_mask
mdd.demo-MDT0000.changelog_mask=
CREAT MKDIR HLINK SLINK MKNOD UNLNK RMDIR RENME RNMTO OPEN CLOSE
LYOUT TRUNC SATTR HSM MTIME CTIME
```

The setting may take a few seconds to update.

3. Register a Changelog user identifier on the MDS and keep a record of the name of the userid that is returned (normally this will be "`cl1`", unless there is more than one Changelog registration):

```
lctl --device <fsname>-<MDT index> changelog_register
```

A complete example follows:

```
[root@m64-1 ~]# lctl --device demo-MDT0000 changelog_register
demo-MDT0000: Registered changelog userid 'cl1'
```

This is not a UNIX user account; it is an identifier used to track Lustre changelog events. Multiple userids can be registered and each will have their own view of the changelog records (Changelogs are stored intelligently, so that there is no duplication of records. However, the changelog register can still become large). Because records take up space on the MDT, be careful to limit the number of registered users. Old Changelog records are purged when each user acknowledges that the record has been read and is no longer required.

Changelog userid registrations are persistent, although they can be deleted (unregistered). The userid cannot be selected at registration: the next available userid in ascending numerical order will be selected and returned. To de-register a userid (and stop changelog processing for that userid), use this command:

```
lctl --device <mdt_device> changelog_deregister <user ID>
```

For example:

```
[root@m64-1 ~]# lctl --device demo-MDT0000 changelog_deregister cl1
demo-MDT0000: Deregistered changelog user 'cl1'
```

Note: `lctl changelog_register` takes no additional arguments (the userid is chosen by Lustre) but you must specify a userid when running `lctl changelog_deregister`.

Prepare the Robinhood Database

Robinhood stores data in a MySQL RDBMS. The database instance must be reachable from the Robinhood policy engine host. For the purposes of this document, it is assumed that the Robinhood server is completely self-contained, hosting both the Robinhood software and the MySQL database instance. Intel® EE for Lustre software will provision Robinhood and the MySQL database onto the same host.

Guidance on the configuration and tuning of MySQL is beyond the scope of this document. Refer to the documentation on the MySQL WWW site (http://mysql.org) for detailed information on establishing scalable and secure MySQL instances. Readers may also find the MySQLTuner project on GitHub helpful (http://github.com/major/MySQLTuner-perl). The Robinhood project mailing list is also a good source of advice on optimization. Refer to the project page for contact information (http://robinhood.sf.net).

To set up a Robinhood database instance:

1. Log into the policy engine host as a user with superuser privileges or SuDO access and ensure that the rbh-config command is installed and in the executable search PATH (normally /usr/sbin):

   ```
   [mjcowe@pe ~]$ rpm -ql robinhood-adm
   /usr/sbin/rbh-config
   [mjcowe@pe ~]$ which rbh-config
   /usr/sbin/rbh-config
   ```

   If these commands fail, then the package is not installed or the PATH environment variable does not include `/usr/sbin`.

2. Start the MySQL daemon:

   ```
   sudo service mysqld start
   ```

3. If this is the first time that the MySQL service has been started, run the secure installation script that is shipped with MySQL:

   ```
   sudo /usr/bin/mysql_secure_installation
   ```

   Accept the defaults and ensure that a password is set for the root account of the database instance. Note that this is the minimum requirement when attempting to protect the database, and is by no means intended to represent a comprehensive solution for database hardening.

4. Ensure that MySQL starts on system boot:

   ```
   sudo chkconfig mysqld on
   ```

5. Verify that the Robinhood database basic prerequisites are satisfied:

   ```
   sudo rbh-config precheck_db
   ```

   This command checks the installation status of some important MySQL commands, and whether or not the server daemon is running. If the server has been installed in accordance with the instructions in this document, the check should pass.

6. Create the Robinhood database:

   ```
   sudo rbh-config create_db
   ```

Without any additional arguments, `rbh-config create_db` will enter an interactive mode and will ask for the following information:

a. A unique identifier for the database instance (use the file system name).

b. A list of hosts that can access the database (this uses SQL formatting, including wildcards, and forms part of the SQL GRANT command; SQL syntax rules apply). For systems where Robinhood and MySQL are co-located, set to `localhost`. Set to `%` to allow any host access to the database.

c. A password for the database user. Robinhood will connect to the database as user `robinhood` with the password you enter here.

d. The root user password for the MySQL database (required in order to be able to create the new database and grant privileges).

7. Verify that the database has been created and that the user `robinhood` can connect:

```
mysql -p -h localhost -u robinhood
...
mysql> show grants;
mysql> use robinhood_<fsname>;
mysql> \q
```

Be careful with the command line syntax: the order of flags is important. A complete example follows:

```
[mjcowe@c64-4p ~]$ mysql -p -h localhost -u robinhood
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.1.69 Source distribution
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All
rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.
mysql> show grants;
+-----------------------------------------------------------------
------------+
| Grants for robinhood@localhost
|
+-----------------------------------------------------------------
------------+
| GRANT SUPER ON *.* TO 'robinhood'@'localhost' IDENTIFIED BY
PASSWORD 'NNN' |
| GRANT ALL PRIVILEGES ON `robinhood_demo`.* TO
'robinhood'@'localhost'    |
+-----------------------------------------------------------------
------------+
2 rows in set (0.00 sec)
mysql> use robinhood_demo;
Database changed
mysql> show tables;
Empty set (0.00 sec)
mysql> \q
Bye
```

8. If no errors are returned, then the user can connect and use the database. There will be no tables inside the database; Robinhood creates these automatically the first time it starts.
9. Record the database password for the user `robinhood` in a file readable by the Robinhood daemon, and make sure that the file is only readable by the process owner. By convention, the password goes into the file `/etc/robinhood.d/.dbpassword`, which is owned by root and with permissions set to 600, although the file location can be changed in the Robinhood configuration.

Robinhood Initial Configuration

The configuration files for Robinhood Lustre HSM are kept in the directory `/etc/robinhood.d/lhsm`. An annotated sample configuration file is supplied as a template with the Robinhood package and can be found in `/etc/robinhood.d/lhsm/templates`. On startup, the init script for Robinhood will scan the configuration directory for any files ending in ".conf" and will create a new Robinhood management instance for each configuration file found. This means that a single Robinhood server might be running a policy engine daemon for several file systems simultaneously. It also means that care must be taken when managing the configuration files to ensure that a single file system is not accidentally managed by two conflicting configurations.

The Robinhood startup script does not traverse subdirectories, so one can keep templates or backup copies of configuration files in a subdirectory of `/etc/robinhood.d/lhsm`, although it is in general preferable to keep development and backup copies of configuration files in entirely separate locations.

Create a Minimal Configuration

The sample configuration template is: `/etc/robinhood.d/lhsm/templates/lhsm_detailed.conf`. If this file is missing, a new one can be generated as follows:

```
rbh-lhsm -T <filename>
```

For example:

```
rbh-lhsm -T /var/tmp/rhtemplate.conf
```

This file can be edited to meet the requirements of the system being configured and then copied to `/etc/robinhood.d/lhsm/`.

The default template is very large and attempts to describe all of the available options for managing a Robinhood policy engine instance. To begin with, create a minimal configuration instead that is sufficient to establish a working database and capture Changelog events. To do this, create a file in `/etc/robinhood.d/lhsm` called `<fsname>-lustre-hsm.conf` (e.g.: `/etc/robinhood.d/lhsm/demo-lustre-hsm.conf`).

The minimum configuration requires four sections:

- `General`
- `Log`
- `ListManager`
- `ChangeLog`

The rest of this chapter shows how to create each section in turn. At the end of the chapter, a complete example configuration is provided.

General

```
General
{
  # file system to be monitored
  fs_path = "/mnt/demo" ;
}
```

The only mandatory field in the `General` section is `fs_path`, which refers to the client mount point of the Lustre file system to be monitored. In this example, a Lustre file system is mounted at `/mnt/demo`.

Log

```
Log
{
  # Log file
  log_file = "/var/log/robinhood/lustre_hsm_demo.log" ;
  # File for reporting purge events
  report_file = "/var/log/robinhood/lustre_hsm_demo_reports.log" ;
  # Alerts file
  alert_file = "/var/log/robinhood/lustre_hsm_demo_alerts.log" ;
}
```

The `Log` section can be left to the default values but it is good practice to have a unique set of log files that are named after the file system being managed by Robinhood.

ListManager (Database)

```
ListManager
{
  commit_behavior = transaction ;
  MySQL
  {
    server = "localhost" ;
    db     = "robinhood_demo" ;
    user   = "robinhood" ;
    password_file = "/etc/robinhood.d/.dbpassword" ;
    engine = InnoDB ;
  }
}
```

The `ListManager` section refers to the database used to record the information that Robinhood gathers. The `commit_behaviour` field can be set to one of `autocommit`, `transaction`, or `periodic`. InnoDB with the `transaction` commit behavior provides better guarantees of consistent data on disk, with the potential penalty of reduced performance.

Note: the Robinhood `ListManager` configuration also references a file that contains the database password. This file must be created before Robinhood starts and must contain a single line with the password in plain text. When creating the file, ensure that the permissions are set so that only the root superuser account can read the content. In the example above, a file called `/etc/robinhood.d/.dbpassword` has been used for this purpose, with the following permissions set:

```
[root@ieel-pe ~]# ls -l /etc/robinhood.d/.dbpassword
-r-------- 1 root root 7 Mar 19 21:52 /etc/robinhood.d/.dbpassword
```

ChangeLog

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

The `ChangeLog` section requires the MDT file system label (normally `MDT0000`) and the `userid` that was returned by Lustre when Changelogs registration was requested (this will normally be `cl1`). The `force_polling` flag is required for Lustre versions 2.0+ because the Changelog readers need to perform active polling to get new events.

Robinhood Initial Configuration – Complete Example

```
General
{
  # file system to be monitored
  fs_path = "/mnt/demo" ;
}
Log
{
  # Log file
  log_file = "/var/log/robinhood/lustre_hsm_demo.log" ;
  # File for reporting purge events
  report_file = "/var/log/robinhood/lustre_hsm_demo_reports.log" ;
  # Alerts file
  alert_file = "/var/log/robinhood/lustre_hsm_demo_alerts.log" ;
}
ListManager
```

```
{
  commit_behavior = transaction ;
  MySQL
  {
    server = "localhost" ;
    db     = "robinhood_demo" ;
    user   = "robinhood" ;
    password_file = "/etc/robinhood.d/.dbpassword" ;
    engine = InnoDB ;
  }
}

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

Starting Robinhood for the First Time

1. Log into the Robinhood policy engine host either as root or as a user that has superuser privileges granted by `sudo`.
2. Mount the Lustre file system.
3. Make sure that the directory for the log files exists before starting Robinhood:

   ```
   sudo mkdir -m 0700 -p /var/log/robinhood
   ```

   If the directory does not exist or is not writable by the Robinhood process, all log entries will be written to `stderr`.
4. Scan the target Lustre file system. This is necessary in order to populate the database with information about files that already exist on the file system. This only needs to be done the first time that Robinhood is started and should ideally be run when the file system is idle (not being used by any other processes):

   ```
   sudo rbh-lhsm --scan --once \
   ```

```
-f /etc/robinhood.d/lhsm/demo-lustre-hsm.conf
```

By default, the `rbh-lhsm` command will select the first configuration file it finds in the `/etc/robinhood.d` directory. If there is more than one configuration, specify the individual file using the `-f` flag, as shown above.

5. Start up the Robinhood service:

```
sudo service robinhood-lhsm start
```

6. Verify that the service is running:

```
service robinhood-lhsm status
```

7. Make sure that the Robinhood service is configured to start automatically on system boot:

```
sudo chkconfig robinhood-lhsm on
```

8. If there are any problems, examine the log files in `/var/log/robinhood`.

Using Robinhood to Create Reports

With the basic configuration described previously, Robinhood does not have any policies defined. However, the monitoring process will still be gathering information and it is possible to interrogate the database and generate reports on the Lustre file system. This chapter gives examples of a few of the commands available to view the information collected by Robinhood. Not all command options are listed, and this chapter is not intended to be a comprehensive manual on every feature of Robinhood. For a full break-down of all the features of the Robinhood command line report tool, run the following command on the policy engine host:

```
rbh-lhsm-report --help
```

All of the commands described in the following sections work whether there are policies defined or not.

Robinhood Daemon Activity

```
sudo rbh-lhsm-report --activity
```

or:

```
sudo rbh-lhsm-report -a
```

For example:

```
[mjcowe@pe ~]$ sudo rbh-lhsm-report -a
```

```
Using config file '/etc/robinhood.d/lhsm/demo-lustre-hsm.conf'.
Filesystem scan activity:

    Previous file system scan:
            start:           2013/07/25 01:00:25
            duration:        41s

    Last file system scan:
            status:          done
            start:           2013/07/25 01:01:22
            end:             2013/07/25 01:02:01
            duration:        39s

         Statistics:
            entries scanned: 11614
            errors:          0
            timeouts:        0
            # threads:       2
            average speed:   318.98 entries/sec

Changelog stats:
        Last read record id:      59750
        Last read record time:    2013/07/21 05:26:28.195000
        Last receive time:        2013/07/25 03:08:46
        Last committed record id: 59749
        Changelog stats:
                 type            total   (diff)    (rate)
                 MARK:               5
                CREAT:           18381
                MKDIR:             331
                HLINK:               0
                SLINK:               0
                MKNOD:               0
                UNLNK:           16381
                RMDIR:             111
                RENME:               0
                RNMTO:               0
```

```
                    OPEN:              0
                   CLOSE:          21461
                   LYOUT:              0
                   TRUNC:              0
                   SATTR:           3080
                   XATTR:              0
                     HSM:              0
                   MTIME:              0
                   CTIME:              0
                   ATIME:              0

Storage usage has never been checked
No migration was performed on this file system
```

If the activity report declares that the file system has never been scanned, re-run the scan step described herein: Starting Robinhood for the First Time.

File System Statistics

High Level Report

```
sudo rbh-lhsm-report --fs-info
```

or:

```
sudo rbh-lhsm-report -i
```

For example:

```
[mjcowe@pe ~]$ sudo rbh-lhsm-report -i

Using config file '/etc/robinhood.d/lhsm/demo-lustre-hsm.conf'.
status    ,     type,      count,     volume,   avg_size
n/a       ,      dir,        333,    2.15 MB,    6.62 KB
new       ,     file,       2000, 1000.98 MB,  512.50 KB
Total: 2333 entries, 1051856896 bytes (1003.13 MB)
```

Identify the Largest File System Consumers

```
# Display largest directories.
rbh-lhsm-report --top-dirs[=<count>], -d <count>
```

```
# Display largest files.
rbh-lhsm-report --top-size[=<count>], -s <count>
# Display oldest entries eligible for purge.
rbh-lhsm-report --top-purge[=<count>], -p <count>
# Display top disk space consumers.
rbh-lhsm-report --top-users[=<count>], -U <count>
```

For example:

```
[mjcowe@pe ~]$ sudo rbh-lhsm-report --top-users

Using config file '/etc/robinhood.d/lhsm/demo-lustre-hsm.conf'.
rank, user      ,   spc_used,      count,   avg_size
   1, root      ,   16.68 GB,       2000,    9.12 MB
```

Detailed Reports

Several options exist for dumping detailed statistics from the Robinhood database:

```
# Dump all file system entries:
rbh-lhsm-report --dump, -D

# Dump all entries for the given user:
rbh-lhsm-report --dump-user <user>

# Dump all entries for the given group:
rbh-lhsm-report --dump-group <group>

# Dump all entries on the given OST.
rbh-lhsm-report --dump-ost <ost_index>

# Dump all entries with the given status
# Status can be one of:
#    unknown
#    new
#    modified|dirty
#    retrieving|restoring
#    archiving
#    synchro
#    released
#    release_pending
rbh-lhsm-report --dump-status <status>
```

For example:

```
[mjcowe@pe ~]$ sudo rbh-lhsm-report --dump
Using config file '/etc/robinhood.d/lhsm/demo-lustre-hsm.conf'.
    type,     status,       size,    user,    group,    migr. class,
purge class,                                   path
...
    file,   released,    1.00 MB,    root,     root,               ,
,       /mnt/demo/nd001/nsd001/nf003
     dir,        n/a,    4.00 KB,    root,     root,               ,
,             /mnt/demo/nd001/nsd001
     dir,        n/a,    4.00 KB,    root,     root,               ,
,             /mnt/demo/nd001/nsd002
     dir,        n/a,    4.00 KB,    root,     root,               ,
,             /mnt/demo/nd001/nsd003
    file,    synchro,    1.00 MB,    root,     root,               ,
,       /mnt/demo/nd001/nsd001/nf001
    file,    synchro,    1.00 MB,    root,     root,               ,
,       /mnt/demo/nd001/nsd001/nf002
    file,        new,    1.00 MB,    root,     root,               ,
,       /mnt/demo/nd001/nsd001/nf004
    file,        new,    1.00 MB,    root,     root,               ,
,       /mnt/demo/nd001/nsd001/nf005
...
```

In the above output, one file is marked `released` and two files are marked as `synchro`. `synchro` means that the archive copy is complete and synchronized with the live copy. Also, two files have not been registered with the HSM (their status is new); no HSM action has been taken for these new files.

Directories are not managed by HSM so their status is marked as `n/a`.

Using Robinhood to Create HSM Policies

Policies are the driving force behind the management of a large file system; the stakeholders of a computing environment define the usage policies and this informs the data management specification incorporated by system managers into the services they implement and support. Policies exist to clearly articulate the requirements of users, owners, and operators of compute and storage platforms.

Lustre itself does not supply a policy management framework, usually referred to as a Policy Engine. This allows system managers and end-users to make their own decisions about how to employ Lustre's HSM technology. However, Robinhood is used as a Policy Engine reference implementation and has an effective configuration language that can be used to translate requirements into enforceable policies.

Robinhood HSM policies fall into three basic categories:

1. Migrate – To copy files to the high capacity archive storage. Migration of a file means to copy the local active file to the remote archive. When a file is archived, two copies exist: one on Lustre, one on the archive platform.
2. Purge – To release files that have been archived in order to free capacity in the Lustre file system. When an archived file is released during a purge of Lustre, the file's data is deleted, freeing up space. The metadata for the file still points to the archive copy.
3. Remove – To clean up archived copies of files deleted from Lustre. Also known as "deferred rm". When a file is removed from Lustre, its archive copy is not deleted immediately. Instead, a remove policy acts like a garbage collector process, cleaning up data marked for deletion asynchronously. Some policies may allow deleted files to remain in the archive for 24 hours, presenting users with an opportunity to "undelete" data that may have been removed by accident.

In addition, Robinhood provides a feature in its configuration language that permits the definition of file classes.

Quick-Start Robinhood Policy Configuration

The configuration example in this section defines a generally useful set of policies applicable to managing a Lustre file system, and represents a good starting point for automating Lustre HSM file management with Robinhood. Explanations for the syntax of this file can be found in subsequent sections. This code block can be added to the base configuration defined earlier for a complete and automated Robinhood installation for managing Lustre HSM file systems.

Note: Robinhood must be restarted whenever the configuration is changed.

```
Filesets {
  FileClass small_files {
    definition {
      tree == "/mnt/demo"
      and
      size <= 1KB
    }
  }
}
```

```
Migration_parameters {
  nb_threads_migration = 4;
  runtime_interval = 15min;
  max_migration_count = 10000;
  max_migration_volume = 10TB;
  check_copy_status_on_startup = TRUE;
}

Migration_policies {
  ignore_fileclass = small_files;

  policy default {
    condition {
      last_mod > 4h
      or
      last_archive > 12h
    }
  }
}

Purge_parameters {
  nb_threads_purge = 4;
  post_purge_df_latency = 1min;
  check_purge_status_on_startup = TRUE;
}

Purge_Policies {
  ignore {
    size == 0
  }

  policy default {
    condition {
      last_access > 12h
      and
      last_mod > 1d
    }
  }
}

Purge_trigger {
  trigger_on         = global_usage;
  high_threshold_pct = 90%;
  low_threshold_pct  = 85%;
  check_interval     = 15min;
}

Purge_trigger {
  trigger_on         = OST_usage;
  high_threshold_pct = 85%;
  low_threshold_pct  = 80%;
  check_interval     = 15min;
}

Purge_trigger {
  trigger_on         = user_usage;
  high_threshold_vol = 1TB;
  low_threshold_vol  = 750GB;
  check_interval     = 4h;
}

hsm_remove_policy {
  hsm_remove = TRUE;
  deferred_remove_delay = 24h;
}

hsm_remove_parameters {
  nb_threads_rm = 4;
  max_rm_count = 10000;
  runtime_interval = 15min;
}
```

General Syntax

Properties are used to create Boolean expressions that define a class of objects. The syntax allows for the creation of complex expressions with unions, intersections, and sub-expressions.

Robinhood supports the following operators:

- equality: ==, !=, <>, >, >=, <, <=
- union: or
- intersection: and
- invert: not
- sub-expressions: ()

and defines the following properties:

- tree
- fullpath
- name
- type
- owner, group
- size
- last_access
- last_mod
- ost_pool
- xattr
- external_command

For example:

```
tree == /mnt/demo/datasetA
and
(owner == mjcowe or owner == johnh)
```

This expression returns `true` if the target exists within the tree `/mnt/demo/datasetA` and the owner is either `mjcowe` or `johnh`.

A complete list of properties supported by Robinhood is outlined next (taken directly from the Robinhood policy engine manual):

- `tree`: entry is under a given path. Shell-like wildcards are allowed.

e.g. `tree == "/fs/subdir/*/dir1"` matches entry `"/fs/subdir/foo/dir1/dir2/foo"`

- `fullpath`: entry exactly matches the path. Shell-like wildcards are allowed. Wildcards will not match the directory path separator ("/").

  `e.g. fullpath == "/fs/*/foo*"` matches entry `"/fs/subdir/foo123"` but it doesn't match `"/fs/subdir/foo4/file"`

- name: entry name matches the given regular expression.

  e.g. name `== "*.log"` matches entry `"/fs/dir/foo/abc.log"`

- `type`: entry has the given type (`directory`, `file`, `symlink`, `chr`, `blk`, `fifo` or `sock`).

  e.g. type == `"symlink"`

- owner or group: entry has the given owner or group (name expected).

  e.g. `owner == "root"`

- `size`: entry has the specified size. You can use suffixes like KB, MB, GB...

- `last_access`: condition based on the last access time of a file (for reading or writing). This is the difference between **current time** and **max(atime, mtime)**. The value can be suffixed with `sec`, `min`, hour, day, `week`, ...

  e.g. `last_access < 1h` matches files that have been read or written within the last hour.

- `last_mod`: condition based on the last modification time to a file. This is the difference between current time and mtime.

  e.g. `last_mod > 1d` matches files that have not been modified for more than a day.

- `ost_pool`: condition about the OST pool name where the file was created. Wild card expressions are allowed.

  e.g. `ost_pool == "pool*"`

- `xattr.xxx.yyy`: test the value of a user-defined extended attribute of the file.

  e.g.: `xattr.user.tag_no_purge == "1"`

- `xattr` values are interpreted as text strings; regular expressions can be used to match `xattr` values. e.g. `xattr.user.foo == "abc.[1-5].*"` would match a file having `xattr user.foo` equal to `"abc.2.xyz"`

  If an extended attribute is not set for a file, it matches empty string. e.g. `xattr.user.foo == "" means xattr "user.foo"` is not defined.

- `external_command`: custom script for testing if an entry matches. Must return 0 if the entry matches, a non-null value otherwise.

N.B. special parameters can be used for when defining the command, e.g.
`external_command( "/usr/bin/do_match {fullpath}" )`

The comment marker is the # symbol. Any characters appearing after the comment marker are ignored.

Filesets – Defining Classes of Files

File classes are all contained within the `Filesets` definition block. This allows administrators to create a shortcut definition for a collection of files that have similar characteristics. Classes can then be referenced in other sections of the policy definition. Each Robinhood policy engine configuration has a single `Filesets` block, containing one or more `FileClass` definitions:

```
Filesets {

  FileClass A {
    Definition {
      ...
    }
  }

  FileClass B {
    ...
  }

  ...
}
```

`FileClass` describes an arbitrary list of properties used to define a set of files. This is commonly used to define to a set of files that belong to a particular user or are stored within a certain directory structure. One of the simplest `FileClass` definitions just refers to anything found under a specific directory tree:

```
FileClass all_lustre_files {
  definition {
    tree == "/mnt/demo"
  }
}
```

This may not be not very useful by itself, so other rules can be added. The following example defines the set of files belonging to an individual user:

```
FileClass mjcowe_lustre_files {
  definition {
    tree == "/mnt/demo"
    and
    owner == "mjcowe"
  }
}
```

Specifying an Archive

A Lustre file system can be served by more than one HSM archive target and FileClass definitions can be associated with specific archives to indicate where to migrate files that match a given definition. Archives may also be specified in the migration policy definitions, which override any archive number referred to in a FileClass. The following example shows how to specify the archive for a FileClass:

```
FileClass mjcowe_lustre_files {
  definition {
    tree == "/mnt/demo"
    and
    owner == "mjcowe"
  }
  archive_id = 1;
}
```

Notes: The `archive_id` property is not nested within the `definition` property. The semi-colon terminating the `archive_id` statement is required. The archive identifier in this statement must be a positive integer greater than zero. If `archive_id` is not specified, the default Lustre archive identifier is used. See Working with HSM Archive Identifiers herein for details regarding the archive identifier.

Robinhood does not support setting the archive identifier to 0 (zero). Archive 0 has a special meaning in Lustre's HSM implementation, representing a "catch-all" archive that will service any archive commands irrespective of the actual archive identifier in the request. This can lead to confusion and use of archive ID 0 is strongly discouraged. Instead, make sure that the Lustre HSM has been configured with a default archive that has a non-zero archive identifier. If Lustre is configured with an archive that has ID 0, do not set an archive ID in Robinhood.

Filesets – A Complete Example

```
Filesets {
  FileClass all_files {
    definition {
      tree == "/mnt/demo"
    }
  }
  FileClass root_files {
    definition {
      tree == "/mnt/demo"
      and
      owner == root
    }
  }
  FileClass user_files {
    definition {
      tree == "/mnt/demo"
      and
      owner != root
    }
  }
  FileClass results_files {
    definition {
      tree == "/mnt/demo/data_sets/results"
    }
  }
  FileClass small_files {
    definition {
      tree == "/mnt/demo"
      and
      size <= 1KB
    }
  }
  FileClass do_not_release {
    definition {
      tree == "/mnt/demo/libraries"
    }
  }
}
```

Migrate – Archiving Files

Migration policies define the sets of files that are to be archived, based on a set of conditions. Migration policies can also optionally specify which HSM archive platform to migrate files to and any files that are to be ignored. The definition of migration policies is optional: if no policy is specified, then files are migrated based on modification time; least-frequently-accessed files are migrated first.

In addition to the policies that define what files to archive, there is a set of global parameters governing data migration in Robinhood. These rules are kept within the `Migration_parameters` block of the configuration file.

Migration Parameters

The following block is an example showing the global migration options for Robinhood HSM:

```
Migration_parameters {
    # numbers of threads used for performing archive requests
    nb_threads_migration = 4;
    # check for files to be copied-out every 15min
    runtime_interval = 15min;
    # Maximum number of copy operations that can be run per pass
    max_migration_count = 10000;
    # Maximum volume of data to archive per pass
    max_migration_volume = 10TB;
    # Archive newly created files that contain no data: TRUE/FALSE
    backup_new_files = FALSE;
    # Check running copies when the daemon restarts: TRUE/FALSE
    check_copy_status_on_startup = TRUE;
    # The amount of time to wait before checking copy status
    # if no feedback has been received
    check_copy_status_delay = 1h;
}
```

Some experimentation is required to establish the correct balance between user requirements and system performance. Consider looking at the average rate-of-change of candidate data, along with the maximum and average file sizes and the amount of time it takes to migrate data to the archive. In the above example, it is assumed that the migration process can safely archive 10TB of data within a 15-minute window, up to a maximum of 10000 requests running across 4 concurrent threads.

Migration Policies

Migration policies are used to specify archiving strategies for one or more sets of files. Each policy sets the conditions under which a set of files is archived and can also optionally tell Robinhood which archive target to use. One can also specify those files that are not to be archived. Policies are evaluated in the order presented in the configuration file. The first policy that matches the file under evaluation is used.

```
Migration_policies {

  ignore {
   ...
  }

  policy <name> {
    target_fileclass = <class>;
    ...
    condition {
      ...
    }
  }

  ...
}
```

The `Migration_policies` block starts with an optional section that lists the set of files to be ignored by the archiving process. Files can be referred to by a specific filter, or they can be listed using a `FileClass` reference. The ignore block uses the same syntax as `FileClass` definitions. For example:

```
ignore {
  size <= 1KB
  and
  owner == mjcowe
}
```

The above statement tells Robinhood not to archive any small files owned by the user `mjcowe`.

`FileClasses` can also be referenced:

```
ignore_fileclass = small_files;
```

Multiple `ignore_fileclass` statements can be included within the `Migration_policies` block.

`Policy` definitions contain a list of `FileClasses` and the conditions under which the files will be archived. For example:

```
policy archive_user_files {
  target_fileclass = user_files;
  condition {
    last_mod > 3h
    or
    last_archive > 12h
  }
}
```

In this example, the policy will apply to any file matching the `user_files` class. Matching files will be migrated to the archive if they have not been modified for at least 3 hours or if they were last archived more than 12 hours ago. The `last_archive` clause covers files that are continuously or frequently updated, and therefore always have a modification time that is lower than the threshold. The `last_archive` clause ensures that these frequently updated files will eventually be copied to the archive.

A default policy can also be defined. This will be applied to any files that do not match any statement in the `ignore` block or `ignore_fileclass` statements, and which do not match the `target_fileclass` statements of the other defined policies. The default policy must appear at the end of the `Migration_policies` block. Any policies or ignore statements that appear after the default definition will be ignored.

```
policy default {
  condition {
    last_mod > 6h
  }
}
```

Migration_policies – Complete Example

```
Migration_policies {


  ignore {
    tree == "/mnt/demo/logs"
```

```
 }
 # Can also use file classes
 ignore_fileclass = small_files;

 policy archive_user_files {
   target_fileclass = user_files;
   condition {
     last_mod > 3h
     or
     last_archive > 12h
   }
 }

 policy high_priority_copy {
   target_fileclass = results_files;
   condition {
     last_mod > 1h
     or
     last_archive > 6h
   }
 }

# Default migration policy
 # Applies to files that don't match previous fileclasses, i.e.:
 #   - don't match the 'ignore' block
 #   - don't match a fileclass of 'ignore_fileclass' directives
 #   - don't match any 'target_fileclass' of migration
 #     policies above
 policy default {
   condition {
     last_mod > 6h
   }
 }
```

```
}
```

Purge – Releasing Files That Have Been Archived

A purge is run whenever the file system exceeds a specified threshold, and is used to free up capacity by releasing previously archived files. Purge definitions have two components: *policies* and *triggers*. A policy defines the set or sets of files that are to be purged and a trigger defines the threshold that will cause a purge event to take place. Triggers can also limit the effect of a purge, for example to a specific user or an OST.

Purges are also governed by a set of global parameters, defined in the `Purge_parameters` block.

Purge Parameters

The following block is an example showing the global purge options for Robinhood HSM:

```
Purge_parameters {
 # numbers of threads used for performing release requests
  nb_threads_purge = 4;
  # Immediately after releasing data, 'df' and 'ost df' may not
  # return exact values, especially if freeing disk space is
  # asynchronous. Thus, it is necessary to wait after a purge
  # before performing 'df' or 'ost df' commands.
  post_purge_df_latency = 1min;
  # check status of previous purge operations on startup
  check_purge_status_on_startup = TRUE;
}
```

This example is taken from the Robinhood default template and is a reasonable starting point for experimentation. The number of running threads will directly impact the responsiveness of Robinhood to purge requests, but must be balanced against the resources available and the number of other actions likely to be executing at the same time.

Purge Policies – Deciding What to Release

A purge policy is used to define the set of archived files to release when a purge event is triggered. Purge events are triggered by capacity constraints on the Lustre file system and are resolved by the policies defined in the `Purge_policies` block. Policies are evaluated in the order presented in the configuration file. The first policy that matches is used.

```
  Purge_policies {
```

```
  ignore {
   ...
  }

  policy <name> {
    target_fileclass = <class>;
    ...
    condition {
      ...
    }
  }

  ...
}
```

The structure of the `Purge_policies` block is very similar to that of the `Migration_policies` block. A section at the start of the block defines a set of files to ignore, followed by a list of policies that determine what files are to be released from the active file system. A final block defines the default policy for any files that do not match the previous entries.

Let's take a look at the ignore block:

```
ignore {
  size == 0
  or
  (
   tree == "/mnt/demo/bin" or tree == "/mnt/demo/sbin"
  )
}
```

In the above example, empty files or files contained within `bin` or `sbin` will not be released when a purge is triggered. One can also use `FileClass` references:

```
ignore_fileclass == do_not_release;
```

Policy definitions contain one or more FileClass targets, followed by a set of conditions under which to release files:

```
policy purge_user_data {

  target_fileclass user_files;
```

```
    condition {
      last_access > 1d
      and
      last_mod > 5d
    }
  }
```

The above policy tells Robinhood to release any user files that have not been accessed within the last day and which have not been modified for at least five days.

The `default` policy can be used to catch any files not previously matched:

```
  policy default {
    condition {
      last_access > 2d
    }
  }
```

Purge_Policies – Complete Example

```
Purge_Policies {

  # files that are never to be released
  ignore {
    size == 0
    or
    (
     tree == "/mnt/demo/bin"
     or
     tree == "mnt/demo/sbin"
    )
  }
  ignore_fileclass == do_not_release;

  policy purge_user_data {
    target_fileclass user_files;
      condition {
        last_access > 1d
        and
        last_mod > 5d
      }
    }

    policy default {
      condition {
        last_access > 2d
      }
    }
  }
```

Purge Triggers – Deciding When to Release Files

Files are released by Robinhood when the Lustre file system exceeds a threshold. These thresholds are defined by one or more "purge triggers" in Robinhood's configuration file. Trigger definitions have a simpler structure and there can be several definitions. There are no default triggers and if no trigger definition exists, the purge policies will never execute.

The following example represents a typical purge trigger:

```
  Purge_trigger {
    trigger_on         = global_usage;
    high_threshold_pct = 85%;
    low_threshold_pct  = 80%;
    check_interval     = 1h;
  }
```

The `trigger_on` parameter indicates scope: any purge policies will be limited to the initial scope defined by this parameter, regardless of any file class definitions in the policies themselves. In other words, `trigger_on` determines the initial scope and the `ignore` and `target_fileclass` definitions inside each purge policy further refine this.

There are four options for `trigger_on`:

1. `global_usage`: trigger is evaluated based on the entire file system.
2. `OST_usage`: capacity is measured relative to individual OSTs. If any single OST exceeds the threshold, then a purge is initiated on that OST, and only that OST. All OSTs are evaluated, but purges only execute on those individual targets that exceed the threshold.

3. `user_usage[(user1[[, user2],...])]`: purge is evaluated against individual users. If the space consumed by an individual user exceeds the threshold, a purge is evaluated against that user's files. Only users that exceed the threshold will have their files purged. One can optionally specify a comma-separated list of users that are to be monitored. This method may be used to enforce a form of quota.
4. `group_usage[(group1[[, group2],...])]`: purge is evaluated against UNIX groups. The syntax and mechanism are equivalent to `user_usage`.

Purges are triggered by start conditions and are concluded by stop conditions and both must be present in the trigger's definition. There can be only one start condition and one stop condition.

The start condition can be one of:

- `high_threshold_pct`: a purge will be triggered when the used capacity of a file system, measured as a percentage, exceeds this value.
- `high_threshold_vol`: a purge will be triggered when the used capacity of a file system, measured as a fixed value in bytes (the default unit of measure), KB, MB, GB or TB, exceeds this value.

The stop condition can be one of:

- `low_threshold_pct`: a purge will stop when the used capacity of a file system, measured as a percentage, is less than or equal to this value.
- `low_threshold_vol`: a purge will stop when the used capacity of a file system, measured as a fixed value in bytes, KB, MB, GB or TB, is less than or equal to this value.

The purge conditions for a file system will be checked by Robinhood at a frequency set by `check_interval`. Each trigger has its own `check_interval` definition.

Purge_trigger – Complete Example

```
Purge_trigger {
  trigger_on         = global_usage;
  high_threshold_pct = 90%;
  low_threshold_pct  = 85%;
  check_interval     = 15min;
}
Purge_trigger {
  trigger_on         = OST_usage;
  high_threshold_pct = 85%;
  low_threshold_pct  = 80%;
  check_interval     = 15min;
}
Purge_trigger {
  trigger_on          = user_usage;
  high_threshold_vol = 1TB;
  low_threshold_vol  = 750GB;
  check_interval     = 4h;
}
```

Remove – Cleaning Up Archive Copies of Deleted Files

When a file is deleted from the active file system, a copy still remains on the archive tier until a cleanup is requested. This is referred to as deferred removal or "soft rm". It is possible to influence the file removal policy using the following configuration block:

```
hsm_remove_policy {
  # enable deferred removal in the archive
  hsm_remove = TRUE;
  # delay before object removal
  deferred_remove_delay = 24h;
}
```

If this block is not set, files are not automatically deleted from the archive, even if they are no longer present on the Lustre file system. File removal is also influenced by another block called `hsm_remove_parameters`:

```
hsm_remove_parameters {
  nb_threads_rm = 4;
  max_rm_count = 10000;
  runtime_interval = 15min;
}
```

This block is very similar to the other parameter blocks:

- `nb_threads_rm`: the number of threads to dedicate to the remove task
- `max_rm_count`: the maximum number of files to delete from the archive in a single pass
- `runtime_interval`: the frequency with which to sweep the archive.

HSM_Remove_Policy & HSM_Remove_Parameters – Complete Example

```
hsm_remove_policy {
  # enable deferred removal in the archive
  hsm_remove = TRUE;
```

```
    # delay before object removal
    deferred_remove_delay = 24h;
  }
  hsm_remove_parameters {
    nb_threads_rm = 4;
    max_rm_count = 10000;
    runtime_interval = 15min;
  }
```

Appendix A: Sample Robinhood Configuration File

```
General {
  # file system to be monitored
  fs_path = "/mnt/demo" ;
}

Log {
  # Log file
  log_file = "/var/log/robinhood/lustre_hsm_demo.log" ;
  # File for reporting purge events
  report_file = "/var/log/robinhood/lustre_hsm_demo_reports.log" ;
  # Alerts file
  alert_file = "/var/log/robinhood/lustre_hsm_demo_alerts.log" ;
}

ListManager {
  commit_behavior = transaction ;
  MySQL {
    server = "localhost" ;
    db     = "robinhood_demo" ;
    user   = "robinhood" ;
    password_file = "/etc/robinhood.d/.dbpassword" ;
    engine = InnoDB ;
  }
}
```

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

Filesets {
  FileClass small_files {
    definition {
      tree == "/mnt/demo"
      and
      size <= 1KB
    }
  }
}

Migration_parameters {
  nb_threads_migration = 4;
  runtime_interval = 15min;
  max_migration_count = 10000;
  max_migration_volume = 10TB;
  check_copy_status_on_startup = TRUE;
}

Migration_policies {
  ignore_fileclass = small_files;

  policy default {
    condition {
      last_mod > 4h
      or
      last_archive > 12h
    }
  }
```

```
}

Purge_parameters {
  nb_threads_purge = 4;
  post_purge_df_latency = 1min;
  check_purge_status_on_startup = TRUE;
}

Purge_Policies {
  ignore {
    size == 0
  }

  policy default {
    condition {
      last_access > 12h
      and
      last_mod > 1d
    }
  }
}

Purge_trigger {
  trigger_on         = global_usage;
  high_threshold_pct = 85%;
  low_threshold_pct  = 80%;
  check_interval     = 15min;
}

Purge_trigger {
  trigger_on         = OST_usage;
  high_threshold_pct = 95%;
  low_threshold_pct  = 90%;
  check_interval     = 15min;
}

hsm_remove_policy {
  hsm_remove = TRUE;
  deferred_remove_delay = 24h;
}

hsm_remove_parameters {
  nb_threads_rm = 4;
  max_rm_count = 10000;
  runtime_interval = 15min;
}
```