MANAGING LUSTRE & ITS DATA @ CEA

LUG Japan | Aurelien Degremont <aurelien.degremont@cea.fr>
CEA, DAM, DIF, F-91297 ARPAJON CEDEX

October 17, 2013
AGENDA

WHAT IS CEA?

LUSTRE ARCHITECTURE

LUSTRE DEVELOPMENTS

- SHINE

- ROBINHOOD

- LUSTRE/HSM BINDING
WHAT IS CEA?

- CEA, Commissariat à l'Energie Atomique et aux Energies Alternatives
  French agency dedicated to research in energy, physics, biology, electronics fields, ...

- CEA is running supercomputers for decades for science simulations.

1963 2012
3 computing centers in few numbers

- Lustre is well known to CEA teams since for 10 years now.
- It is used in production on clusters since 2003.
- Moving to dedicated scratch filesystems to complex, center-wide, data management relying on a full Lustre infrastructure.

**TERA**
- 3 supercomputers
- More than 1 PFlops
- More than 5000 Lustre nodes
- 500 GB/s of Lustre bandwidth
- 23+ PB of Lustre filesystems
- 12 Lustre filesystems

**TGCC/CCRT**
- 2 supercomputers
- More than 6200 Lustre nodes
- 250 GB/s of total Lustre bandwidth
- 12+ PB of Lustre filesystems
- 5 Lustre filesystems
Scratch data are local to clusters
Simulation results are directly written to a central filesystem
- Zero copy data access for post-processing clusters
- Directly connected to HSM
- Automatic and transparent migration between Lustre and HSM
- Lustre as a very big cache in front of the HSM
LUSTRE DEVELOPMENTS

SHINE
Shine is an open source Python-based tool

- CEA open source project, since 2007, in Python (v2.4+).
- Latest version 1.3, freely available on SourceForge:
  - http://lustre-shine.sf.net/

Objectives:
- Hide Lustre complexity. Do not need to be a Lustre expert to administrate it!
  - Throw away all your Lustre wrapper scripts
- Highly scalable to meet big CEA Lustre filesystem constraints.
  - Rely on ClusterShell for efficient command execution
  - Run on 5000 clients and 1000 OSTs in few seconds

- Based on a CLI (admins like working with terminals)
  - Customizable
  - Aggregate results and display consolidated outputs

- Validated on Lustre version 1.8 to 2.4
SHINE: ARCHITECTURE

Setup
- Shine is deployed on management and all Lustre nodes (Only 2 RPMs)
- Shine heavily relies on your existing SSH infrastructure.
  - No complex communication daemons! No key! No additional config files!
- Shine replicates filesystem configuration on all filesystem nodes

Interface
- Admins control the filesystem through a central point of management
  - Shine will connect to required nodes transparently
- Or run locally on remote node for local actions only.
Lustre filesystem components are described in a configuration file called a *model*. This model should include:

- **File system name**
  
  ```
  fs_name: tokyo
  ```

- **NID/node mapping**
  
  ```
  nid_map: nodes=nova[2-5] nids=nova[2-5]@tcp0
  ```

- **Device per target type**
  
  ```
  # MGS
mgt: node=nova2 dev=/dev/sde1

  # MDT
mdt: node=nova3 dev=/dev/sdf

  # OST
  ost: node=nova4 ha_node=nova5 dev=/dev/mapper/lun[1-6]
opt: node=nova5 ha_node=nova4 dev=/dev/mapper/lun[6-11]
  ```

- **Clients and mount path**
  
  ```
  client: node=nova[10-19]
mount_path: /mnt/lad2012
  ```

And that's sufficient!
## SHINE: USAGE EXAMPLES

### Format

- No issue with MGS NIDs or failover NIDs.

```
# shine format -f tokyo
Format tokyo on nova[2-5]: are you sure? (y)es/(N)o: y
Starting format of 14 targets on nova[2-5]

<table>
<thead>
<tr>
<th>TYPE</th>
<th>#</th>
<th>STATUS</th>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGT</td>
<td>1</td>
<td>offline</td>
<td>nova2</td>
</tr>
<tr>
<td>MDT</td>
<td>1</td>
<td>offline</td>
<td>nova3</td>
</tr>
<tr>
<td>OST</td>
<td>12</td>
<td>offline</td>
<td>nova[4-5]</td>
</tr>
</tbody>
</table>
```

### Status

- With a recovery in progress on 1 OST for 5 clients

```
# shine status -f tokyo -x sicknode

<table>
<thead>
<tr>
<th>TYPE</th>
<th>#</th>
<th>STATUS</th>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGT</td>
<td>1</td>
<td>online</td>
<td>nova2</td>
</tr>
<tr>
<td>MDT</td>
<td>1</td>
<td>online</td>
<td>nova3</td>
</tr>
<tr>
<td>OST</td>
<td>11</td>
<td>online</td>
<td>nova[4-5]</td>
</tr>
<tr>
<td>OST</td>
<td>1</td>
<td>recovering for 99s (0/5)</td>
<td>nova5</td>
</tr>
<tr>
<td>CLI</td>
<td>5</td>
<td>mounted (recovering=1)</td>
<td>nova[10-14]</td>
</tr>
<tr>
<td>CLI</td>
<td>5</td>
<td>mounted</td>
<td>nova[15-19]</td>
</tr>
</tbody>
</table>
```
Lots of other features not detailed here

- Display
  - Consolidate views
  - High control on display
- Lustre components
  - Routers support (start, stop and status)
  - Client-only or MGS-only filesystems
- Lustre tunings and configurations
  - External journal device
  - Default striping
  - Format options
  - Mount options
  - Mount path
  - Multirail: Multiple NIDs per server
  - Eviction detections
  - Quota
  - Tunefs
- And more...
Swiss army knife for your filesystem

- Policy Engine and reporting tool for large filesystem
- Open Source product developed by CEA
  - http://robinhood.sf.net/
- Current version is 2.4.3, but next release, 2.5, adds new features and performances.

- Monitor filesystem activities by scanning or reading events (Lustre Changelogs)
- Save all metadata in a (My)SQL database
- Query this database at will to:
  - Audit, accounting, alerts
  - Migration, purge
  - Everything based on policies
- Thanks to CLI or WebGUI

- Use to control good usages and keep your filesystem healthy
Feeding the database

- Robinhood information and actions are based upon the database data.
- Robinhood supports MySQL as backend.

- Database could be filled using:

  - Parallel filesystem scan
    - For Lustre 1.8 or any POSIX filesystem.
  
  - Reading Lustre Changelog
    - For Lustre 2.x
    - Only an initial scan is needed.

  Parallel scan
  (nightly, weekly, …)

Lustre v2
ChangeLogs

Robinhood database

near real-time
DB update
Fast *find* and *du* clones

- Query Robinhood DB instead of performing POSIX namespace scan ➔ faster!

```bash
> rbh-find [path] -user "foo*" -size +1G -ost 4
20sec for 40M entries
```

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

```bash
> rbh-du -sH /fs/dir -u foo --details /fs/dir
    symlink count:30777, size:1.0M, spc_used:9.1M
    dir      count:598024, size:2.4G, spc_used:2.4G
    file     count:3093601, size:3.2T, spc_used:2.9T
```
ROBINHOOD: FINE-GRAINED STATISTICS

Top users and groups
- Sorted by volume, object count, avg file size...

```bash
> rbh-report --top-users --by-count
```

<table>
<thead>
<tr>
<th>rank</th>
<th>user</th>
<th>spc_used</th>
<th>count</th>
<th>avg_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>john</td>
<td>423.23 GB</td>
<td>1599881</td>
<td>275.30 KB</td>
</tr>
<tr>
<td>2</td>
<td>paul</td>
<td>292.91 GB</td>
<td>954153</td>
<td>330.98 KB</td>
</tr>
<tr>
<td>3</td>
<td>mike</td>
<td>65.37 GB</td>
<td>543169</td>
<td>130.98 KB</td>
</tr>
</tbody>
</table>

...

Top directories
- Sorted by object count, avg file size...

```bash
> rbh-report --top-dirs --by-count
```

<table>
<thead>
<tr>
<th>rank</th>
<th>path</th>
<th>dircount</th>
<th>avgsize</th>
<th>user</th>
<th>group</th>
<th>last_mod</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/hpss/foo1/dir1</td>
<td>24832</td>
<td>2.62 GB</td>
<td>foo1</td>
<td>gr59</td>
<td>2013/03/11 17:13:45</td>
</tr>
<tr>
<td>2</td>
<td>/hpss/foo2/dir3</td>
<td>20484</td>
<td>339.88 MB</td>
<td>foo2</td>
<td>g03</td>
<td>2013/02/03 06:59:05</td>
</tr>
<tr>
<td>3</td>
<td>/hpss/bar2/dir4</td>
<td>19484</td>
<td>543.82 MB</td>
<td>bar2</td>
<td>g03</td>
<td>2012/05/28 12:45:26</td>
</tr>
</tbody>
</table>

...
### Web GUI

#### Space used per group

<table>
<thead>
<tr>
<th>Group</th>
<th>Space used</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra1</td>
<td>544.09 TB</td>
<td>12,708,782</td>
</tr>
<tr>
<td>pa0</td>
<td>302.19 TB</td>
<td>2,743,617</td>
</tr>
<tr>
<td>gen2</td>
<td>166.26 TB</td>
<td>3,370,537</td>
</tr>
<tr>
<td>ra05</td>
<td>150.26 TB</td>
<td>1,995,581</td>
</tr>
<tr>
<td>ra12</td>
<td>82.22 TB</td>
<td>1,393,682</td>
</tr>
<tr>
<td>ra2</td>
<td>79.94 TB</td>
<td>7,465,108</td>
</tr>
<tr>
<td>ra07</td>
<td>73.46 TB</td>
<td>522,402</td>
</tr>
</tbody>
</table>
File size profiling

- Available in the Robinhood web interface
- File size repartition

Global / per file size

Summary per user

<table>
<thead>
<tr>
<th>User</th>
<th>Total volume</th>
<th>File count</th>
<th>Avg file size</th>
<th>file size ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>empty</td>
</tr>
<tr>
<td>11.98 TB</td>
<td>36 251</td>
<td>346.48 MB</td>
<td>0.74%</td>
<td>3.17%</td>
</tr>
<tr>
<td>15.02 TB</td>
<td>86 584</td>
<td>181.94 MB</td>
<td>21.73%</td>
<td>36.00%</td>
</tr>
<tr>
<td>74.17 TB</td>
<td>38 685</td>
<td>1.96 GB</td>
<td>16.62%</td>
<td>22.65%</td>
</tr>
<tr>
<td>27.31 TB</td>
<td>65 944</td>
<td>434.31 MB</td>
<td>21.16%</td>
<td>30.35%</td>
</tr>
<tr>
<td>57.35 TB</td>
<td>112 730</td>
<td>533.41 MB</td>
<td>1.97%</td>
<td>6.21%</td>
</tr>
</tbody>
</table>

User guidance

- Informational emails sent on specific criteria
- Avg. file size < 50MB, more than 80% of files < 32MB
- Includes good I/O practices, stats about their account (list of non-compliant directories)
Our dashboard

robinhood info

volume

inodes

fs1

fs2

fs3

bandwidth

MDS RPCs

# of clients

column view allows event correlation for each filesystem
FS TEMPERATURE: USER WORKING SETS

Time-lapse of filesystem usage

- working set = set of files recently written/read

Data production (mod. time)
- 70% of data produced within the last 2 months

Data in use (last access)
- 80% of data accessed <1 month
- 20% of data accessed >1 month

70% of data produced within the last 2 months

1 month working set

2 months working set

“Cold” data

80% of data accessed <1 month

Read bursts

<table>
<thead>
<tr>
<th>volume modified</th>
<th>cumul.</th>
<th>volume accessed</th>
<th>cumul. (neg. values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h-15m</td>
<td>155.6GB</td>
<td>00h-15m</td>
<td>983.6GB</td>
</tr>
<tr>
<td>15m-01h</td>
<td>450.1GB</td>
<td>15m-01h</td>
<td>4.0TB</td>
</tr>
<tr>
<td>01h-06h</td>
<td>1.3TB</td>
<td>01h-06h</td>
<td>12.1TB</td>
</tr>
<tr>
<td>06h-12h</td>
<td>3.6TB</td>
<td>06h-12h</td>
<td>15.7TB</td>
</tr>
<tr>
<td>12h-18h</td>
<td>8.4TB</td>
<td>12h-18h</td>
<td>22.3TB</td>
</tr>
<tr>
<td>18h-24h</td>
<td>22.3TB</td>
<td>18h-24h</td>
<td>256.0TB</td>
</tr>
<tr>
<td>24h-30h</td>
<td>464.3TB</td>
<td>24h-30h</td>
<td>464.3TB</td>
</tr>
<tr>
<td>30h-36h</td>
<td>682.9TB</td>
<td>30h-36h</td>
<td>855.8TB</td>
</tr>
<tr>
<td>36h-42h</td>
<td>886.9TB</td>
<td>36h-42h</td>
<td>16.0TB</td>
</tr>
<tr>
<td>42h-48h</td>
<td>522.3TB</td>
<td>42h-48h</td>
<td>24.5TB</td>
</tr>
<tr>
<td>&gt;48h</td>
<td>2.5PB</td>
<td>&gt;48h</td>
<td>2.5PB</td>
</tr>
<tr>
<td>Total</td>
<td>2.5PB</td>
<td>Total</td>
<td>2.5PB</td>
</tr>
</tbody>
</table>
Visualization of different filesystem usage patterns

- Significant reads/writes (volume)
- Cooling effect after a large read (volume)
- Read-mostly filesystem (volume)
- Robinhood DB dump and initial scan (inodes)
  Nice linear scan, ~1.5 days for 50M inodes
Fileclasses based on file attributes

Admin defined rules
Policy definition:
- Flexible and highly customizable
- Attribute-based
- Using fileclass definitions

```plaintext
Fileclass system_log_files {
    definition {
        name == "*.log" and
        (owner == "root" or group == "root")
    }
}
Fileclass big_pictures {
    definition {
        (name == "*.png" or name == "*.jpg")
        and (size > 10MB)
    }
}
Fileclass flagged {
    definition {
        xattr.user.flag == "expected value"
    }
}
```

Get class summary

```plaintext
$ rbh-report --classinfo
class , count, spc_used, volume, min_size, max_size, avg_size
Documents , 128965, 227.35 TB, 250.69 TB, 8.00 KB, 2.00 TB, 30.03 GB
System_log_files , 1536, 4.06 TB, 4.06 TB, 684, 200.01 GB, 2.71 GB
Big_pictures , 621623, 637.99 TB, 638.02 TB, 3, 1.54 TB, 1.05 GB
```
ROBINHOOD POLICIES

Apply policies to fileclasses

- Several built-in policies
  - Purge (for scratch filesystem)
  - Directory removal
  - Deferred removal (for undelete)
  - Backup and Archiving
  - HSM: schedule *archiving* and *release*

- Examples:

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

purge_policies {
  ignore_fileclass = my_fileclass;
  policy purge_logs {
    target_fileclass = BigLogFiles;
    condition { last_mod > 15d } } }
```
LUSTRE DEVELOPMENTS

LUSTRE/HSM BINDING
A long-awaited project!

- A CEA project started several years ago.
- It has known all Lustre companies.
- After lots of modifications and rewrites, it is finally there!

It is landed!

- Thanks to Intel, the whole code is now landed
- Partially landed in Lustre 2.4
- Has reached total inclusion in Lustre 2.5
- Will be available in it, at the end of October 2013, which will be the next maintenance branch

- Currently under test and debugging
Principle

- HSM seamless integration

Take the best of each world:

- **Lustre**: High performant disk-cache in front of the HSM
  - Parallel filesystem
  - High I/O performance
  - POSIX access

- **HSM**: long term data storage
  - Manage large number of cheaper disks and tapes
  - Huge storage capacity

Ideal for center-wide Lustre filesystem.
Features

- Migrate data to HSM (*Archive*)
- Free disk space when needed (*Release*)
- Bring back data on cache-miss (*Restore*)

- Policy management (migration, purge, soft removal, …)
- Import from existing backend
- Disaster recovery (restore Lustre filesystem from backend)

New components

- Copy tool (backend specific user-space daemon)
- Policy Engine (user-space daemon)
- Coordinator
New components: Coordinator, Agent and Copy tool

- The coordinator gathers archive requests and dispatches them to agents.
- Agent is a client which runs a copytool to transfer data between Lustre and the HSM.
PolicyEngine manages Archive and Release policies

- A user-space tool which communicates with the MDT and the coordinator.
- Watches the filesystem changes.
- Triggers actions like `archive`, `release` and removal in backend.
**COMPONENTS**

Copytool

- It is the interface between Lustre and the HSM.
- It reads and writes data between them. It is HSM specific.
- It runs on a standard Lustre client (called Agent).

- 2 of them are already available:
  - **POSIX** copytool. Could be used with any system supporting a POSIX interface.
    - It is provided with Lustre
  - **HPSS** copytool. (HPSS 7.3.2+).
    - CEA development which will be freely available to all HPSS sites.

- More supported HSM to come:
  - **DMF** (SGI)
  - **OpenArchive** (GRAU DATA)
Example RobinHood policy: Migration

Migrate files older than 12 hours with a different behavior for small ones.

```plaintext
Filesets {
  FileClass small_files {
    definition { tree == "/mnt/lustre/project" and size < 1MB }
    migration_hints = "cos=12" ;
    ...
  }
}

Migration_Policies {
  ignore { size == 0 or xattr.user.no_copy == 1 }
  ignore { tree == "/mnt/lustre/logs" and name == "*.log" }

  policy migrate_small {
    target_fileclass = small_files;
    condition { last_mod > 6h or last_archive > 1d }
  }
  ...
  policy default {
    condition { last_mod > 12h }
    migration_hints = "cos=42" ;
  }
}
```
CONCLUSION
CEA has a long history of Lustre usage. It has developed a deep knowledge of it.

- This is useful for a good system administration
- And helps to develop tools and patches for Lustre

File system patterns and volumes are closely watched. Tools are developed to:

- Watch filesystem usages
- Advice user of better practices
- Remove and migrate data to optimize filesystem usage
- Understand real user needs and estimate future storage needs
Thanks.

Questions?