SHINE
Open source Lustre administration tool
AGENDA

Objectives
What is it?
Architecture
First filesystem
Features
  · Status
  · Tuning
  · Update
  · And more...
Performance & Scalability
Future releases
OBJECTIVES

- Help using Lustre without requiring deep Lustre knowledge
  - Lustre commands are not so easy
  - Tuning also
- Necessary to write your own scripts to wrap Lustre commands

- Shine development is driven by the following targets:
  - Manage Lustre filesystem life cycle
  - Simple and easy to use
  - Fast and scalable
WHAT IS IT?

Shine is an open source Python-based tool

- Licensed under GPL v2
- Distributed model
  - Is executed on management node and remote nodes
- Event based
- Website
  - http://lustre-shine.sf.net/

Requirements

- Python 2.4 to 2.7
- ClusterShell 1.5.1+
  - Rely on it for command execution
- Tested on RHEL 5 & 6 and Fedora 12+
- Support any Lustre version starting from 1.6, including 2.x
WHAT IS IT?

- **Preparation**
  - Lustre RPMs should be installed
  - LNET should be set up correctly
  - Target devices should be usable

- **Installation**
  - 2 RPMs on each node
    - shine
    - clustershell
  - SSH setup on each node, without password
    - Heavily rely on your existing SSH infrastructure
  - Nothing else!
    - No complex communication daemons
    - No configuration files, no key
**ARCHITECTURE**

- **Setup**
  - Shine is deployed on management and all Lustre nodes
  - 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.
FIRST FILESYSTEM
Demo cluster configuration with:
- 1 management node
- 4 I/O servers with dedicated storage drives
- 10 Lustre clients
Model file

- Lustre filesystem components are described in a configuration file called a *model*.
- This model should include:
  - File system name
    
    ```
    fs_name: lad
    ```
  - 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]
    ost: node=nova5 ha_node=nova4 dev=/dev/mapper/lun[7-12]
    ```
  - Clients and mount path
    
    ```
    client: node=nova[10-19]
    mount_path: /mnt/lad2012
    ```

- And that's sufficient!
Install the model and use it!

Install it to copy the model file as configuration file where needed.

```
# shine install -m /etc/shine/models/lad.lmf
Using Lustre model file /etc/shine/models/lad.lmf
Configuration files for file system `lad' have been installed successfully on nova[2-5,10-19]
Lustre targets summary:
  1 MGT on nova2
  1 MDT on nova3
  12 OST on nova[4-5]
Use `shine format -f lad' to initialize the file system.
```

Format

No issue with MGS NIDs or failover NIDs.

```
# shine format -f lad
Format lad on nova[2-5]: are you sure? (y)es/(N)o: y
Starting format of 14 targets on nova[2-5]
FILESYSTEM STATUS (lad)
  TYPE  # STATUS NODES
    ----  - ------  -----  
      MGT   1 offline nova2
      MDT   1 offline nova3
      OST  12 offline nova[4-5]
```
Start everything!

- Start the server part
  - It takes care of starting in right order and OST in parallel

  ```
  # shine start -f lad
  Starting 12 targets of lad on nova[2-5]
  Start successful.
  
  FILESYSTEM STATUS (lad)
  TYPE  # STATUS NODES
  ----  - ------ -----
  MGT   1 online nova2
  MDT   1 online nova3
  OST   12 online nova[4-5]
  ```

- Client can now be mounted
  - Also started in parallel

  ```
  # shine mount -f lad
  Starting lad clients on nova[10-19]...
  Mount successful on nova[10-19]
  ```
High control on display

Display consolidate and compact view of filesystem status

```
# shine status -f lad
= FILESYSTEM STATUS (lad) =
  TYPE  # STATUS  NODES
    ----  - ------  -----  
     MGT   1 online  nova2
     MDT   1 online  nova3
     OST  12 online  nova[4-5]
     CLI  10 mounted nova[10-19]
```

But highly configurable

- Based on predefined views

```
# shine status -t ost -i 1-2 -V disk
======================
FILESYSTEM DISKS (lad) =========================

<table>
<thead>
<tr>
<th>DEVICE</th>
<th>SERVERS</th>
<th>DEV_SIZE</th>
<th>TYPE</th>
<th>INDEX</th>
<th>LABEL</th>
<th>FLAGS</th>
<th>FSNAME</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sde2 nova4      8.0TB OST   1 lad-OST0001       lad    online</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/dev/sde3 nova4      8.0TB OST   2 lad-OST0002       lad    online</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Custom format, for scripting by example
  - Extract OST nodes from configuration

```
# shine config -f lad -t ost -H -O '%nodes'
nova[4-5]
```
The current Lustre way to managed tunings, \texttt{lctl conf_param}, has drawbacks.
  
  - Difficult to list and change them
  - Tunings are lost when doing a writeconf

Central point to define Lustre tuning: in a simple config file.
Shine can tune /proc values according to a configuration file, depending on:
  
  - Node name
  - Node type (MGS, MDS, OSS, CLT)

### ALIAS DECLARATION ###

\begin{verbatim}
alias debug=/proc/sys/lnet/debug
alias max_rpcs_in_flight=/proc/fs/lustre/osc/*/${ost}*/max_rpcs_in_flight
alias max_dirty_mb=/proc/fs/lustre/osc/*/${ost}*/max_dirty_mb
alias statahead_max=/proc/fs/lustre/llite/*/statahead_max
\end{verbatim}

### TUNING PARAMETERS ###

\begin{verbatim}
"0" debug CLT;OSS;MDS;nova[120-137]
"32" max_rpcs_in_flight CLT
"64" max_dirty_mb CLT
"0" statahead_max CLT
\end{verbatim}
Updating an existing filesystem could be tricky

Shine detects filesystem configuration changes and proposes the command to be run.

- If you want to add a new OST by example:
  - Update your model file
    
    ```
    ost: node=nova6 dev=/dev/mapper/lun[1-5]
    ```
  - Run update command:
    
    ```sh
    # shine update -m /etc/shine/models/lad.lmf
    Using Lustre model file ./lad.lmf
    FILESYSTEM CHANGES
    Format: 6 component(s) on nova6
    Start: 6 component(s) on nova6
    Update `lad': do you want to continue? (y)es/(N)o: y
    Updating file system configuration file `lad.xmf' on nova6
    NEXT ACTIONS (should be done manually)
    >You can now `format' 6 new target(s)
    > shine format -f myfs -l lad-OST[0030-0035]
    >You can now `start' 6 new component(s)
    > shine start -f myfs -l lad-OST[0030-0035]
    Update is finished.
    ```
  - Check output and run the proposed commands
Lots of other features not detailed here

- Multirail: Multiple NIDs per server
- Routers start, stop and status
- Tunefs
- Eviction detections
- External journal device
- Quota
- Client-only or MGS-only filesystems
- Default striping
- Format options
- Mount options
- Mount path

And more...
Designed for small to very large systems

- Configuration files are kept simple
  - Even with lots of OSTs: 3 OSS in failover with a third of 48 OSTs each.
    
    ```
    ost: node=nova4 ha_node=nova5 ha_node=nova6 dev=/devmapper/lun[1-48/3] index=[0-15]
    ost: node=nova5 ha_node=nova6 ha_node=nova4 dev=/devmapper/lun[2-48/3] index=[16-31]
    ost: node=nova6 ha_node=nova4 ha_node=nova5 dev=/devmapper/lun[3-48/3] index=[32-47]
    ```

  - Or thousands of clients
    
    ```
    client: node=nova[1000-3500] mount_path=/mnt/fs1
    client: node=nova[5000-6500] mount_path=/mnt/fs2
    ```

- Shine is running a lot of Lustre commands in parallel.
  - ClusterShell is used for that and it has already shown very good performances.
    - http://clustershell.sf.net/ (OLS 2012 paper)
  - Few numbers (from TERA-100):
    - Checking status of ~800 OSTs, on ~50 OSS: 2 sec
    - Checking status of ~450 servers: 2.2 sec
    - Checking status of ~3600 busy clients: 50 sec
    - 11 PB filesystem, on ~800 OSTs, fsck'd in: 45 min
Next releases will be focused on...

- Parallelism
  - Filesystems will be managed in parallel
- Lustre modules loading and unloading
- Health check
- Tuning
  - Better error handling
  - Applied more efficiently
- Failover
  - Automatic detection of migrated targets
Thank you!

Questions?