
whamcloud

whamcloud

European Lustre
Workshop 2011

Improving Lustre management with Hydra

John Spray
john@whamcloud.com

whamcloud

Contents

whamcloud

Introduction to Hydra

- 'Hydra' is the codename for Whamcloud's forthcoming Lustre management product
- The objective is to simplify provisioning, management and monitoring of Lustre filesystems
- This presentation covers some of the technical approaches we have taken to the challenge of improving Lustre administration.

Components

- Web interface (GUI)
- CLI, allowing scripting
- Web service, which is the basis for the GUI+CLI
- Database, storing all configuration information and historical metrics
- Worker process pool, running actions remotely on the Lustre servers
- Vendor-specific plugins providing direct communication with storage controllers.

Technologies used

- Python
- Django
- MySQL (or existing SQL database server)
- RabbitMQ (a popular messaging framework)
- Celery (a distributed worker framework)

Theme: exploiting general purpose technology wherever possible to keep the core Lustre-specific code lean and mean.

whamcloud

Communicating with Lustre servers

- We install an agent executable on Lustre servers
- Sites may favor different protocols:
  - HTTPS (either agent-as-host or hydra-as-host)
  - SSH
  - Existing agent infrastructure
- When outgoing connections are required, we use a generic worker pool (celery):
  - Based on standards (AMQP) and capable of being run in a distributed environment
  - Supports multiprocessing or event-driven concurrency
  - Makes it easy for us to support different communications methods
- Lustre servers “phoning home” to Hydra are simply handled by Apache.

Gathering statistics

- We started out evaluating LMT, but:
  - It became one of the least user-friendly parts of the setup process
  - Adding/changing stats in C was inconvenient compared with python
- We could have coped with it, but we also wanted to be able to inline stats within our existing agent communications
  - Inlining stats with agent communications takes advantage of existing network setup and is adequate for small-medium installations
  - Retain the option of an out-of-band binary UDP channel (like cerebro, ganglia) for largest systems
- So in a typical configuration, we receive updated stats along with the periodic status update the agent sends to the server.

Visualizing statistics

- Many generic monitoring tools provide very limited graphing: “You can have any chart you want as long as it’s a line chart”
- We recognize the challenges of monitoring 100s or 1000s of block devices: it isn’t helpful to put 1000 lines on a line graph:
  - Sometimes need very high density display, such as heatmaps
  - Sometimes certain metrics together, for example the MDT IOPS breakdown by operation.
  - Sometimes data are not simple scalar time series, e.g. latency histogram or IO size histogram
  - Sometimes we can’t know in advance which statistics will be interesting, so graphs must be customizable

Controlling the cluster

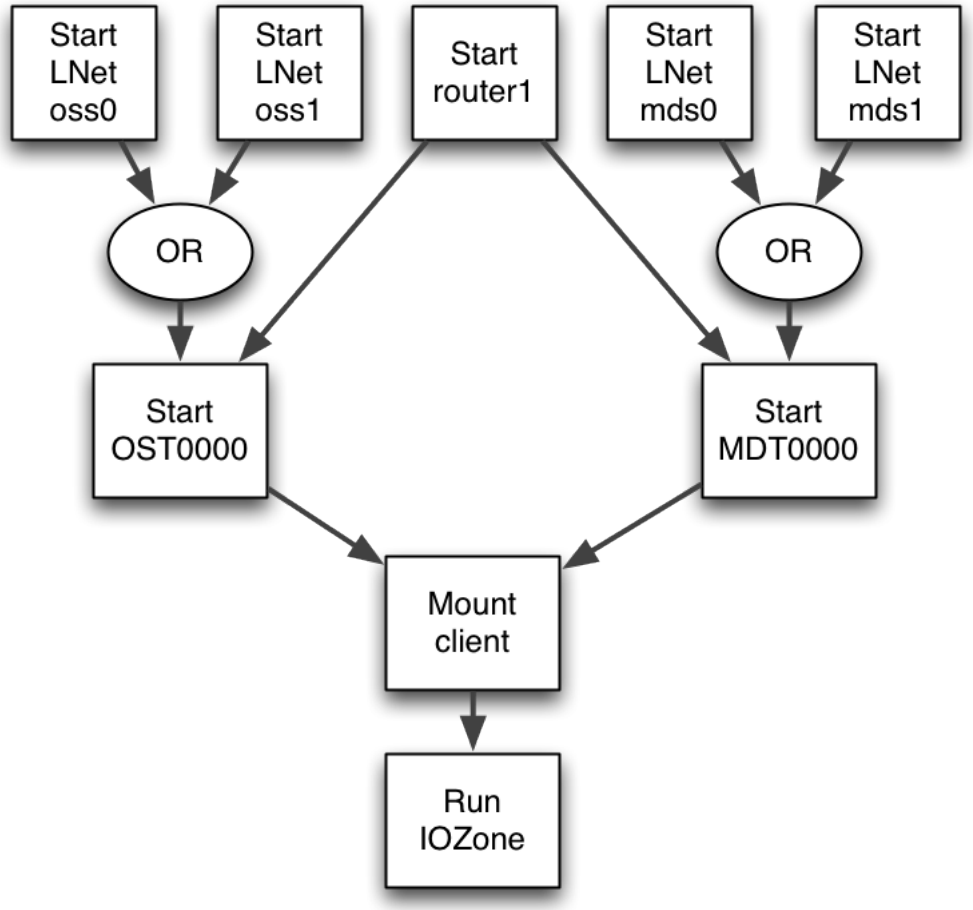
- It is not as simple as “putting a GUI on it”, because there is nothing central on which to put a GUI.
- Just wrapping “mount” doesn't solve the usability problem: it can still stall or fail without telling you why.
- As there is no existing central point of control for Lustre systems, we must create one which knows about the system elements and how they interact.

What does ‘start filesystem’ mean?

- If this is the first mount, ensure the MGS is up
- Ensure routers are up
- Start all the OSTs and the MDT
- For each target, one or more of its servers must be up and have LNet started

- We must run steps in parallel where possible, and stop if a dependency cannot be satisfied.
- We must ensure that this overall task cannot be interfered with by other user actions, while allowing the user to run other actions which do not overlap with it.

Example: running a benchmark



Additional benefits of dependency-oriented approach

- If e.g. a target can't start because a router is failing to start, we can explain to the user exactly what is wrong, rather than simply failing to mount.
- Because we implement even basic operations like formatting and starting a target as dependency-aware actions, we have created a framework into which more complex actions will easily fit:

Storage hardware integration

- Lack of integration is burdensome:
  - Provisioning: Administrators have to manually select devices for formatting and decide failover configuration
  - Monitoring: No quick way to distinguish storage issues from hardware issues, or understand how storage issues affect a filesystem.
- But integration is challenging:
  - Different vendors provide different management APIs/CLIs.
  - Different devices provide different levels of information or types of error
  - Between the hardware and Lustre there are OS layers (LVM, multipath)

Stack from Lustre to disks

- Goal is to convert "Disk 0xab87e4f is close to failure" to "MDT0000 has a failing disk"
- To do this, we need a full record of the relationships between entities in Hydra.

Storage plugins

- Plugins are written in Python
- Plugins describe 'resources' (e.g. a disk, an enclosure, a fan) and their relationships
  - Resources have arbitrary attributes (some vendors might report RPM for a fan, some might not)
  - Resources can report arbitrary statistics
  - The graph of relationships is essential: it is what lets us map a problem with a disk to an affected filesystem.
- Hydra generates UI for plugins based on their declared resources, relationships and attributes, but some vendors may choose to implement custom UI pages in addition to their plugin.

Plugins simplify provisioning

- Information from storage controllers allows us to make provisioning extremely simple for users.
- We now know about paths (and preferred paths), so the user selects a LUN by its name on the controller.
- …of course, we will still support setting these things manually for power users.

Plugin API

- Important to make this simple to ease adoption
- Allow flexible degree of implementation:
  - Basic: detecting configured storage resources at installation time
  - Monitoring: more complete plugins for reporting status and metrics
  - Full: plugins capable of configuring storage from bare metal
- Simple syntax:

```
1 class LvmGroup(base_resources.StoragePool):
2     identifier = GlobalId('uuid')
3
4     uuid = attributes.Uuid()
5     name = attributes.String()
6     size = attributes.Bytes()
```

Conclusion

- Hydra brings together smart approaches to provisioning, management and monitoring to build a new Lustre management platform
  - Flexible acquisition of filesystem status and metrics
  - Manage actions within a dependency-aware cluster model
  - A plugin system for storage hardware
- We showed a prototype at ISC11, the next demonstration of Hydra will be at SC11 in November.
- General availability will follow in the spring.


whamcloud