LUSTRE USAGE MONITORING
What the & #@ are users doing with my filesystem?

Kilian CAVALOTTI, Thomas LEIBOVICI | CEA/DAM

LAD’13 – SEPTEMBER 16-17, 2013
Lustre monitoring is hard
- very few tools are available
- “distributed filesystem” means “distributed stats”

Administrators need stats
- to check that everything runs smoothly
- to analyze problems
- to watch users
  - users are (sometimes unintentionally) malicious, they need to be watched
- because stats are cool
  - and graphs are even cooler

We @CEA/DAM needed tools
- to monitor filesystem activity (performance, trends...)
- to understand filesystem usage (typology, distribution, working sets...)

LAD’13 - SEPTEMBER 16-17, 2013 | PAGE 2
LUSTRE MONITORING: WHAT WE WANT
Monitoring filesystem activity

- tools
- visualizing real-time activity
  - bandwidth, metadata operations, space usage...
- diagnosing slowdowns and unexpected behavior
  - suggesting improvements for user applications causing slowdowns
- understanding how people use their files
  - showing the filesystem “temperature” by visualizing working sets and their evolution

Answering typical questions

- “I don't get the expected GB/sec…”
- “My ls takes too long!”
- “Can you give me the current usage for each user working on this project?”
- “What kind of data is in the filesystem?”

Disclaimer

- we use Lustre 2.1. (YMMV...)
MONITORING FILESYSTEM ACTIVITY
The Lustre admin tool belt

- ltop: github.com/chaos/lmt
  top-like command providing instant stats about a filesystem
- collectl: collectl.sourceforge.net
  sar-like tool for the monitoring of Infiniband traffic, Lustre activity, etc.
- htop, dstat and friends

Open-source tools developed @CEA

- shine: lustre-shine.sourceforge.net
  command-line tool for Lustre filesystem configuration and management
- robinhood policy engine: github.com/cea-hpc/robinhood
  multi-purpose tool for large filesystem management

Custom scripts

- based on all of the above, glue provided by Bash and Python
  we parse /proc/fs/lustre a lot
- nice colors courtesy of RRDtool
Swiss-army knife for your filesystems

- audit, accounting, alerts, migration, purge, based on policies
- massively parallel
- stores filesystem metadata in a (My)SQL database
  - allows lightning-fast usage reports
  - provides rbh-du and rbh-find replacements for du and find
- advanced capabilities for Lustre filesystems
  - purge on OST usage, list files per OST
  - pool and/or OST based policies
  - changelogs reader (Lustre v2): no scan required
- reporting tools (CLI), web interface (GUI)
- open-source license

@CEA

- one instance per Lustre filesystem, using changelogs
Our dashboard

- Column view allows event correlation for each filesystem
- Robinhood info
- Volume
- Inodes
- Fs1 bandwidth
- Fs2 MDS RPCs
- Fs3 # of clients
**Bandwidth**

/proc/fs/lustre/obdfilter/\${fsname}\-OST*/stats

- **snapshot_time**: 1377780180.134516 secs.usecs
- **read_bytes**: 178268 samples [bytes] 0 1048576 185094526864
- **write_bytes**: 220770 samples [bytes] 115 1048576 230859704715
- **get_page**: 390038 samples [usec] 0 6976 69863795 15160124841
- **cache_access**: 45189234 samples [pages] 1 1 45189234
- **cache_miss**: 45189234 samples [pages] 1 1 45189234
- **get_info**: 5250 samples [reqs]
- **set_info_async**: 25 samples [reqs]
- **process_config**: 1 samples [reqs]
- **connect**: 6074 samples [reqs]
- **disconnect**: 12 samples [reqs]
- **statfs**: 181816 samples [reqs]
- **create**: 30 samples [reqs]
- **destroy**: 520 samples [reqs]
- **setattr**: 76 samples [reqs]
- **ping**: 10349451 samples [reqs]

Aggregated on each OST, for each filesystem.
Metadata operations (MDS RPCs)

- `/proc/fs/lustre/mdt/${fsname}-MDT0000/mdt*/stats`
- most commonly used:
  - `obd_ping`: filesystem health
  - `mds_(dis)connect`: clients (u)mounts
  - `mds_get(x)attr`: metadata operations
  - `mds_statfs`: (lfs) `df`
  - `mds_readpage`: readdir()
  - `ldlm_ibits_enqueue/mds_close`: ~open()/close()

Long-term storage filesystem: large files, few metadata operations

2 filesystems with the same clients (~6000), but different usages

Scratch-like filesystem: lots of open()
From graph to clients

peak detection on the RPC graph
initially a manual process: watching the graph
now automatic: averaging the last 10 values in the RRD DB, and comparing to a threshold

collection of RPC samples
initially a manual process: enabling debug traces, collecting, disabling debug traces
now automatic: background collection for 5sec every 5min, keeping 7 days of traces

echo +rpctrace > /proc/sys/lnet/debug
echo +rpc > /proc/sys/lnet/subsystem_debug

analysis of collected samples
RPC type breakdown by Lustre client

<table>
<thead>
<tr>
<th>NODENAME</th>
<th>NET</th>
<th>COUNT</th>
<th>%</th>
<th>CUMUL</th>
<th>DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.100.10.137</td>
<td>o2ib6</td>
<td>18567</td>
<td>21.07</td>
<td>21.07</td>
<td>mds_getxattr: 18567</td>
</tr>
<tr>
<td>10.100.10.138</td>
<td>o2ib6</td>
<td>17534</td>
<td>19.90</td>
<td>40.97</td>
<td>mds_getxattr: 17534</td>
</tr>
<tr>
<td>10.100.10.140</td>
<td>o2ib6</td>
<td>14724</td>
<td>16.71</td>
<td>57.69</td>
<td>mds_getxattr: 14724</td>
</tr>
</tbody>
</table>

mds_getxattr peak
3 nodes involved
From client identification to code improvement

querying the cluster scheduler

```bash
$ sacct -S startdate -E endate -N nodename
```

going getting jobs running on those nodes during the event

contacting the user

for live events, possibility to `strace` the process at fault

having seen such things as:

```bash
# strace -e stat -p 95095 -r
Process 95095 attached - interrupt to quit
0.000000 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000398 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000246 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.038258 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000338 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000275 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.038236 stat("/path/to/case/WORKDIR/DMP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000320 stat("/path/to/case/WORKDIR/BF", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
0.000234 stat("/path/to/case/WORKDIR/STOP", 0x7fff0c3ce630) = -1 ENOENT (No such file or directory)
```

kindly suggested the user better ways to stop his run
Time-lapse of filesystem usage

- working set = set of files recently written/read

70% of data produced within the last 2 months

Data production (mod. time)

Data in use (last access)

80% of data accessed <1 month

80% of data accessed <1 month

"Cold" data

Read bursts

1 month working set

2 months working set

Filesystem working set (volume) - store - 1 month

<table>
<thead>
<tr>
<th>Volume modified</th>
<th>Cumul.</th>
<th>Volume accessed</th>
<th>Cumul. (neg values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00s-15m</td>
<td>155.6GB</td>
<td>155.6GB</td>
<td>00s-15m 983.6GB 983.6GB</td>
</tr>
<tr>
<td>15m-01h</td>
<td>450.1GB</td>
<td>605.7GB</td>
<td>15m-01h 4.0TB 4.0TB</td>
</tr>
<tr>
<td>01h-06h</td>
<td>1.3TB</td>
<td>1.3TB</td>
<td>01h-06h 12.1TB 12.1TB</td>
</tr>
<tr>
<td>06h-01d</td>
<td>3.6TB</td>
<td>4.8TB</td>
<td>06h-01d 15.7TB 15.7TB</td>
</tr>
<tr>
<td>01d-07d</td>
<td>56.7TB</td>
<td>64.5TB</td>
<td>01d-07d 223.2TB 223.2TB</td>
</tr>
<tr>
<td>07d-30d</td>
<td>399.8TB</td>
<td>464.3TB</td>
<td>07d-30d 1.3PB 1.3PB</td>
</tr>
<tr>
<td>30d-60d</td>
<td>662.9TB</td>
<td>1.1PB</td>
<td>30d-60d 855.8TB 2.4PB</td>
</tr>
<tr>
<td>60d-90d</td>
<td>860.9TB</td>
<td>1.9PB</td>
<td>60d-90d 16.0TB 2.4PB</td>
</tr>
<tr>
<td>&gt;90d</td>
<td>522.3TB</td>
<td>2.5PB</td>
<td>&gt;90d 24.5TB 2.5PB</td>
</tr>
</tbody>
</table>

Total: 2.5PB

Total: 2.5PB
Implementation

- compute \{\text{modification, access}\} age for each file
  \[
  \text{now - last_mod/last_access}
  \]
- aggregate count/volume by age range
- based on Robinhood database
  - optimized SQL query
  - could be slow for filesystems with > 100s millions of inodes
- some RRDtool magic involved
  - the RRDtool graph command line is \sim 6500\text{ characters long...}

Benefits

- visualization of the current working set
  - age repartition, based on last modification and last access dates
- allows to anticipate filesystem evolutions
  - mostly recently written data: need to add more OSTs
  - mostly old, never-accessed files: could be purged
Visualization of different filesystem usage patterns

- **significant reads/writes (volume)**
  - ![Graph](image1.png)

- **cooling effect after a large read (volume)**
  - ![Graph](image2.png)

- **read-mostly filesystem (volume)**
  - ![Graph](image3.png)

- **Robinhood DB dump and initial scan (inodes)**
  - nice linear scan, ~2 days for 50M inodes
  - ![Graph](image4.png)
ANSWERING TYPICAL QUESTIONS

a.k.a.
THE ROBINHOOD COOKBOOK
Question

“I don't get the expected GB/sec...”

Different filesystems, different usages

- user documentation indicates recommended file sizes for each filesystem as well as purge policies, quotas...
- often, recommendations not followed → bad performance
- usually a case of “small files”
  - writing large blocks is more efficient
  - metadata overhead decreases performance

Implemented solutions

- filesystem quotas
  - a limited amount of inodes motivates users to write bigger files
- user guidance
  - file size profiling, small file detection using Robinhood
  - information provided to users if necessary, helping them to take appropriate measures
File size profiling
- available in the Robinhood web interface
- file size repartition
global / per user

summary per user

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)
- graduated response system
  first two emails are warnings
  new job submission is suspended after the 3rd warning
  without any action or feedback from the user, data is automatically packed
**Robinhoood reports**

- file size profiling
  - get a list of users sorted by the ratio of files in a given size range

  ```
  $ rbh-report --top-users --by-szratio=1..31M
  ```

<table>
<thead>
<tr>
<th>rank</th>
<th>user</th>
<th>spc_used</th>
<th>count</th>
<th>avg_size</th>
<th>ratio(1..31M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>john</td>
<td>1.23 TB</td>
<td>26763</td>
<td>60.59 MB</td>
<td>91.30%</td>
</tr>
<tr>
<td>2</td>
<td>perrez</td>
<td>80.77 GB</td>
<td>27133</td>
<td>114.71 MB</td>
<td>87.64%</td>
</tr>
<tr>
<td>3</td>
<td>matthieh</td>
<td>2.22 TB</td>
<td>25556</td>
<td>1.04 GB</td>
<td>85.76%</td>
</tr>
<tr>
<td>4</td>
<td>vladimir</td>
<td>62 GB</td>
<td>14846</td>
<td>74.18 MB</td>
<td>78.50%</td>
</tr>
<tr>
<td>5</td>
<td>gino</td>
<td>30.92 GB</td>
<td>15615</td>
<td>77.34 MB</td>
<td>77.02%</td>
</tr>
</tbody>
</table>

  - tip: with “-l FULL”, Robinhood logs SQL requests, for reuse and customization

- small file detection
  - list directories with at least 500 entries, sorted by average file size (smallest first)

  ```
  $ rbh-report --top dirs --by-avgsize --reverse --count-min=500
  ```
Question
“*My ls takes too long!*”

Issue
- directories with too many entries (100,000s)
- sequential processing of readdir() takes a long time

Implemented solutions
- real-time directory alerts
- alerts defined in Robinhood configuration

```bash
Alert large_dir { type == directory
    and dircount > 10000
    and last_mod < 1d }
```

⚠️ doesn’t work with changelogs, only when scanning

- report directories with the most entries

```bash
$ rbh-report --topdirs
```
Question

“Can you give me the current usage for each user working on this project?”
given that:

- some data are in a project directory,
- users have some other data from the same project in their own directory,
- project data can sometimes be identified by a group attribute, and be anywhere else ...

Issues

- data belonging to a single project can be distributed in several locations
- logical attachment to a project can reside in file attributes, not in location
**Implemented solution**

- basic accounting using rbh-report
- space used in the whole filesystem by user (splitted by group)

```
$ rbh-report -u foo* -S
```

```
user , group, type, count, spc_used, avg_size
foo1 , proj001, file, 422367, 71.01 GB, 335.54 KB
...
Total: 498230 entries, 77918785024 bytes used (72.57 GB)
```

for a specific group

```
$ rbh-report -g proj001
```

- per directory accounting using rbh-du filtering capabilities by a given user in a given directory

```
$ rbh-du -u theuser /project_dir
```

by a given group in given directories

```
$ rbh-du -g proj001 /users/foo*
```

- project data in other locations

```
$ rbh-find -g proj001
```
Question

“What kind of data is in my filesystem?”

Implemented solution

- define Robinhood file classes
- arbitrary definitions based on file attributes
  - name, owner, group, size, extended attributes...

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

$ rbh-report --classinfo

<table>
<thead>
<tr>
<th>Class</th>
<th>Count</th>
<th>SPC Used</th>
<th>Volume</th>
<th>Min Size</th>
<th>Max Size</th>
<th>Avg Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>128965</td>
<td>227.35 TB</td>
<td>250.69 TB</td>
<td>8.00 KB</td>
<td>2.00 TB</td>
<td>30.03 GB</td>
</tr>
<tr>
<td>System_log_files</td>
<td>1536</td>
<td>4.06 TB</td>
<td>4.06 TB</td>
<td>684</td>
<td>200.01 GB</td>
<td>2.71 GB</td>
</tr>
<tr>
<td>Big_pictures</td>
<td>621623</td>
<td>637.99 TB</td>
<td>638.02 TB</td>
<td>3</td>
<td>1.54 TB</td>
<td>1.05 GB</td>
</tr>
</tbody>
</table>
WRAPPING UP
CONCLUSION

Visualizing filesystem activity is useful

- we can diagnose abnormal patterns and improve suboptimal code
- we can better understand user behavior and help them getting the best I/O performance out of their applications

Solutions exist

- some assembly required
- but all the required information is there in /proc/fs/lustre in the filesystem metadata (Robinhood database)
- you need Robinhood and some RRDtool-fu
THANK YOU!

QUESTIONS?