> Version 1.5 Q1 2008

LECTURE 4.2 Log Analysis

Content

- Goal
- Generating logs
- Understanding logs
- Log analysis tools
- Lock dumps
- Stack traces
- Crash/gdb - mcore/netdump

Goal

How to understand Lustre

- Can read source code
  - But the effect of VFS on behavior is hard to anticipate
- Using debug logs
  - Contains enough information to understand a lot
  - Is difficult
  - Increasingly during normal operation, no debug logs
    - It affects performance pretty badly

How to debug Lustre

- During development:
  - Use logs, use (k)gdb
- During production use:
  - Console output
  - Lock dumps
  - Request dumps
  - Stack traces
  - Crash with netdump or with mcore
- For very nasty problems:
  - Use light weight tracing
  - See source & LustreDebugging on wiki

Log generation

Dumping logs

- The kernel has a 5MB buffer.
  - This is, in fact, not nearly as much as it seems.
  - `/proc/sys/portals/debug_mb`
- A mask can be set:
  - `/proc/sys/portals/debug`
- Then subsystems can be (de)selected:
  - `/proc/sys/portals/subsystem_debug`
- The dump location is also controllable:
  - `/proc/sys/portals/debug_path`
  - Default `/tmp/lustre-log-localhost.localdomain`

Getting a Debug Log

- Sometimes the system volunteers a debug log.
  - > After some kernel Oopses, and all Lustre LBUGs
- Other times, we'll ask you to generate one.
- If we do, please clear the buffers before you reproduce the debug log using:
  - > lctl clear

Post-processing

- If you get a log the normal way...
  - lctl debug_kernel [filename]

  ...then lctl will post-process it for you.

- If the kernel dumps it on its own (i.e., an LBUG):
  - It will contain binary information (pointers to text strings)
  - Process this with:
    lctl debug_file <infile> <outfile>
- Please do this before you send it to us.

Dumps

- A debug daemon can write the logs continuously.
  - This has been useful in several cases
- The Lustre wiki has a page about how to start and stop the daemon.

Understanding the DEBUG log

Inode bits lock DEBUG message

- 00010000:00010000:0:1151031337.272617:640:8791:0:
- Subsystem:mask:cpu:time-sec.usec:stack:pid:ext_pid:
- (ldlm_lockd.c:1100:ldlm_handle_bl_callback())
- (file:line no:function)
- ### already unused, calling callback (e0c4275d)
- Free form message
- ns: mds-mds1_UUID lock: c3987d80/0x511830a47d84b222
- ns:namespace lock:ptr/local handle
- lrc: 2/0,0 mode: CR/CR res: 31257/3224802362
- lrc: lockrefs/rdrs,wrtrs mode: granted/reqtd, res[1]/res[2] here ino/gen
- bits 0x2 rrc: 1 type: IBT flags: 4010
- Bits: 0x2 rrc:res refc, type: i-bit lock, flags: CB_XXX
- remote: 0x0 expref: -99 pid 9783
- remote handle, exportref: <unused> pid: last thread having lock

Lock Bits

- /* INODE LOCK PARTS */
- #define MDS_INODELOCK_LOOKUP 0x000001 /* dentry, mode, owner, group, acls, stripe ea */
- #define MDS_INODELOCK_UPDATE 0x000002 /* size, links, timestamps */
- #define MDS_INODELOCK_OPEN   0x000004 /* For opened files */

Lock flags

- #define LDLM_FL_CBPENDING   0x000010
  - /* this lock is being destroyed */
- #define LDLM_FL_LOCAL   0x004000
  - /* local lock (ie, no srv/cli split) */

Extent lock line

```
00010000:00010000:0:1151098795.206458:4736:15549:0:
(ldlm_request.c:507:ldlm_cli_enqueue())
### client-side enqueue START
ns: OSC_lin-cli1.cfs_ost9_MNT lock:
   c6db4d80/0x8464f70ca019676e
lrc: 3/1,0 mode: --/PR res: 6025/0 rrc: 1 type: EXT
```

- Above is the same as before, note that no lock is granted yet

```
[0->18446744073709551615] (req 0->18446744073709551615)
```

- Offered / Requested extent; this is the EOF lock

```
flags: 0 remote: 0x0 expref: -99 pid: 15549
```

- Same as before

RPC DEBUG line

00000100:00100000:0:1151097766.922030:2560:10953:0:(service.c:618:ptlrpc_server_handle_request())

- Same as before

Handling RPC

- There is also:
  - Handling, Handled (server),
  - Sending, Completed (client)

pname:cluuid+ref:pid:xid:nid:opc
ll_ost_01:e89_lov1_d7d+2:11943:711921:12345-0@lo:400

- Handling/requesting process, client uuid — ref:process:xid:nidpid:opc

Log analysis tools

llanalyze.pl

- A compact 300 line tool
  - Indent and color logs
  - Extract features (e.g. locks, RPCs, one PID)
  - Can relate calls among multiple logs to show RPC patterns
- llanalyze.pl needs a maintainer

- llvisualize
  - Written by people from Intel
  - Very pretty output
  - An order of magnitude bigger than llanalyze
  - Probably in complete disrepair

Lock dumps

Lockdump

--- Namespace: OSC_lin-cli1.cfs_ost12_MNT (rc: 3, client: 1)

> Locks granted by the OST to this OSC, refcount, ???

--- Resource: cdf3dd80 (6164/0/0/0)

> Pointer, (object id/0/0/0) an extent in this namespace

Granted locks:

-- Lock dump: c6db4b80/0x8464f70ca0196783 (rc: 1)

> Lock pointer and local handle

Node: NID 0@lo (rhandle: 0x8464f70ca01967c2)

> Lock servers nid, and the handle of the lock there

Resource: cdf3dd80 (6164/0)

> Back pointer from lock to the resource

Req mode: PR, grant mode: PR, rc: 1, read: 0, write: 0 flags: 0x100000

Extent: 0 -> 18446744073709551615 (req 0-18446744073709551615)

> Already discussed above

Stack traces

Stack traces

```
ll_mdt_rdpg_0 S 00000023  6484 11935    1   11936 11934 (L-TLB)
```

> What thread is this a stack of?

```
c2a3ff5c 00000046 e0c9613e 00000023 00000282 c2c82c30 0006ddc8 c010ae46
c2c82c30 00000000 c1405740 c1404de0 00000000 00003f3a 55784afa 0000cc68
c2c82c30 d50945b0 d509471c 00000000 c2a3ff80 ffffffff ffffffff 00000282
```

> This is register information, not used often

```
Call Trace:
 [<e0c9613e>] ptlrpc_server_free_request+0x20/0x1cc [ptlrpc]
 [<c010ae46>] do_gettimeofday+0x1a/0x9c
 [<e0c9914a>] ptlrpc_main+0x853/0xb79 [ptlrpc]
 [<c011d6d3>] default_wake_function+0x0/0xc
 [<e0c988ea>] ptlrpc_retry_rqbds+0x0/0xd [ptlrpc]
 [<c02d113a>] ret_from_fork+0x6/0x14
 [<e0c988ea>] ptlrpc_retry_rqbds+0x0/0xd [ptlrpc]
 [<e0c988f7>] ptlrpc_main+0x0/0xb79 [ptlrpc]
 [<c01041f5>] kernel_thread_helper+0x5/0xb
```

> As is commonly seen this stack trace is not 100% correct. This thread is almost certainly waiting instead!

Generating traces - SysRq

- Sometimes the system does it for you
  - Oops, LBUG, watchdog timers
- Sysrq
  - **/etc/sysctl.conf**, add **kernel.sysrq=1**
  - Operate with: **echo t > /proc/sysrq-trigger**
- SysRq-P (one stack trace) is usually uninteresting
- SysRq-T (all stack traces) is voluminous but very useful
  - Especially if a process is hung and wont make progress
- SysRq-M (memory info) is sometimes enlightening
  - Is the system essentially out of memory?
  - Are any of the counters impossible values?

crash/gdb – mcore/netdump

Crash

- Is a gdb extension with very convenient macros
  - Macros can easily show all file handles etc.
    - By hand this takes time
- Crash can operate on
  - A live kernel
  - An mcore dump – compact, very reliable, on the node
  - A netdump – similar to mcore, over the wire, less reliable

Other gdb debugging techniques

- Using kgdb with
  - > VMware
  - > Physical serial ports
  - > Ethernet – less reliable
- On the whole, kgdb is excellent for development

- Use gdb with UML
  - > UML is often difficult to get running
  - > Debugging is extremely convenient

THANK YOU