FROM RESEARCH TO INDUSTRY

Lustre Maleficarum:

At scale bug hunting using VMs

Lustre Admins &
Developers Workshop
2016

Dominique Martinet <dominique.martinet@cea.fr>

www.cea.fr

21 SEPTEMBER 2016

Plan-ish

Lustre debugging

- Illustrating tools with a real problem
  - pcocc: CEA-made large scale virtualization
  - logs (dmesg, lctl dk)
  - crash (live/kdump)
  - gdb (qemu)
  - systemtap
- Closure on our problem

Foreword

Caution: Not safe for post-lunch brain

- A lot of detailled examples we will NOT get into
  - Will explain some details, but not all
  - Can come back on slides I skipped in questions
- Meant to be able to read the slides later a second, third time and be useful later
  - Complex command examples

DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Tools: PCoCC
Private Cloud on Compute Cluster

pcocc – Private Cloud On Compute Cluster

CEA development

- Helper to start any number of VM (one to thousands) on production clusters
  - We have HPC resources,
  - Need to follow restrictions
- Open source (please hit François Diakhaté <francois.diakhate@cea.fr> until he publishes it, or just ask him to provide you with a copy of the code)
- Known usages:
  - Continuous Integration
    - nfs-ganesha: spawn isolated server/clients and run tests on every gerrit changeset
  - Live test platform for interns/contractors
    - Lustre development
    - Scalability tests (1000 client VMs start to stress things a bit)
    - Prepare maintenances and test updates

pcocc – Private Cloud On Compute Cluster

More specifically

- Manages VMs in a slurm job
  - VM images can be different, initialisation with cloud-init too.
  - slurm limitation that all VMs must have the same resources (cpu/mem)
- Images backed on lustre directly for performance
  - Cannot access compute resources directly from VMs, but qemu can!
- Provides IB with SRIOV and manages opensm configuration to isolate each cluster on their own pkey
- Similar isolation for ethernet network (openvswitch)

As simple as that (after initial setup)

```
$ pcocc alloc -t 00:30:00 -c 7 -p broadwell mds0,oss0,oss1,scs5:129
salloc: Pending job allocation 43169
salloc: job 43169 queued and waiting for resources
salloc: job 43169 has been allocated resources
salloc: Granted job allocation 43169
Configuring hosts... (done)
$ pcocc ssh -t vm0 "sudo shine start"
Start successful.
= FILESYSTEM STATUS (testfs0) =
TYPE # STATUS NODES
---- - ------ -----
MGT  1 online vm0
MDT  1 online vm0
OST  4 online vm[1-2]
$ pcocc ssh -t vm0 "sudo shine mount"
[12:20] In progress for 129 component(s) on vm[3-131] ...
testfs0 was successfully mounted on vm[3-131]
= FILESYSTEM STATUS (testfs0) =
TYPE # STATUS  NODES
---- - ------  -----
CLI  129 mounted vm[3-131]
$ pcocc ssh vm3 sudo touch /ccc/testfs/testfile
$ pcocc ssh vm3 clush -bw vm[3-131] ls /ccc/testfs/
---------------
vm[3-131] (129)
---------------
testfile
```

Useful commands

- Reset multiple VMs at once
  - clush -R exec -w vm[4,67,123] pcocc reset %h
- Get a console to watch boot/crash progress
  - pcocc console vm0
- Save changes (qemu-guest-agent will freeze/thaw filesystems)
  - pcocc save [-d $CCCSCRATCHDIR/pcocc/newimage] vm3
- Raw qemu “human” monitor commands:
  - pcocc monitor-cmd vm0 gdbserver tcp::1234
  - pcocc monitor-cmd vm0 nmi # crash (sysctl kernel.unknown_nmi_panic)

Further reading

- Please see the developer's presentation for more details (performance, etc)

  http://www.teratec.eu/library/pdf/forum/2016/Presentations/A1_03_Forum_TERATEC_2016_DIAKHATE_CEA.pdf

DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

A first glance at our problem

An actual hang

Observed behavior

- MDS mostly unresponsive
  - for all clients, only obd pings worked, but totally idle (0 load)
- Had to crash/restart it
  - killing identified clients was not enough
  - Sometimes gets stuck again after restart...

Recipe

- Lustre servers (production in 2.5, reproduced with 2.7 too)
- Lots of clients (2.7)
  - Some open (RW/O_CREAT) files in a directory
  - Most access (stat) said directory

An actual hang

Reproducer

- When you get all the computing stuff out of the way...
- Can have fun with xargs ;-)

```
mkdir “/ccc/testfs/userdir/foo.d”

clush -bw vm[3-130] '
  seq 0 1000 | \
    xargs -P 7 -I{} sh -c "
      (({}%3==0)) && \
        touch /ccc/testfs/userdir/foo.d/foo$(hostname -s | tr -d vm) || \
        stat /ccc/testfs/userdir/foo.d > /dev/null"'
```

(Okay, we used an MPI version of that for finer control)

DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Tools: logs

Lustre logs

First thing to check: log messages

- dmesg gets important logs by default (warning error emerg console)
- lctl dk: in memory log buffer
  - Can change debug level (lctl get_param debug)
    - trace inode super ext2 malloc cache info ioctl neterror net warning buffs other dentry nettrace page dlmtrace error emerg ha rpctrace vfstrace reada mmap config console quota sec lfsck hsm
  - Can change debug subsystem (lctl get_param subsystem_debug)
    - undefined mdc mds osc ost class log llite rpc mgmt lnet lnd pinger filter echo ldlm lov lquota osd lfsck lmv sec gss mgc mgs fid fld
  - Can change the size (lctl set_param debug_mb)

Sample line from dmesg

```
[670569.918144] LustreError: 0:0:(ldlm_lockd.c:343:waiting_locks_callback()) ### lock callback timer expired after 151s:
evicting client at 10.2.3.4@o2ib1  ns: mdt-myfs0-MDT0000_UUID lock: ffff880fa069c4c0/0x425687c61f16399b lrc: 3/0,0 mode:
PR/PR res: [0x200006031:0xc3b4:0x0].0 bits 0x13 rrc: 368 type: IBT flags: 0x60200400000020 nid: 10.2.3.4@o2ib1 remote:
0xcf5cd9f5b05921bb expref: 6 pid: 771 timeout: 4965569040 lvb_type: 0
```

DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Tools: crash

Gathering information and crash dump analysis

- List processes
  - process state (UN (likely hang), RU (running), IN (waiting)...)
  - pipe to grep, awk, output to file...
  - list process open files (*files* command)
- Get backtraces to look for similar JIRA LU#
- Dig further and get useful structures out of backtraces
  - ptlrpc request peer
  - ldlm_resource fid
  - anything in memory! But usually on post-mortem analysis
- Automatic dumps, always helpful:
  - “service”: kdump
  - settings: panic_on_lbug, unknown_nmi_panic, etc

crash – sample outputs - ps

Back to our hang, on the MDS...

```
crash> ps -l | grep -E 'mdt0[0-9]_'
[6705942128390 38] [IN]  PID: 11372  TASK: ffff880fa0769540  CPU: 3   COMMAND: "mdt01_041"
[6705942128054 50] [IN]  PID: 11454  TASK: ffff880fcb367500  CPU: 11  COMMAND: "mdt01_062"
[6705919032037 36] [IN]  PID: 11350  TASK: ffff880ff484aae0  CPU: 12  COMMAND: "mdt02_035"
[skipped 13]
[6705899776771 63] [IN]  PID: 11466  TASK: ffff880f9fae9500  CPU: 3   COMMAND: "mdt01_065"
[6705699588598 12] [IN]  PID: 11247  TASK: ffff88106eb4c040  CPU: 2   COMMAND: "mdt01_012"
[6705679710964 80] [IN]  PID: 4611   TASK: ffff8810737caaa0  CPU: 11  COMMAND: "mdt01_009"
[6704191206995 15] [IN]  PID: 4636   TASK: ffff881072ed2aa0  CPU: 7   COMMAND: "mdt03_009"
[6704190340448 07] [IN]  PID: 771    TASK: ffff881026cf4ae0  CPU: 11  COMMAND: "mdt01_002"
[6704170464150 03] [IN]  PID: 11501  TASK: ffff880f9f4d8080  CPU: 4   COMMAND: "mdt02_074"
[6704170463733 74] [IN]  PID: 11514  TASK: ffff880fc03fd500  CPU: 6   COMMAND: "mdt03_079"
[6702661286724 54] [IN]  PID: 11480  TASK: ffff880fc2142080  CPU: 14  COMMAND: "mdt03_067"
[skipped 12]
[6702661106676 18] [IN]  PID: 11465  TASK: ffff880fc593a080  CPU: 0   COMMAND: "mdt00_062"
[6701152157395 87] [IN]  PID: 11505  TASK: ffff880f9f583540  CPU: 14  COMMAND: "mdt03_075"
[6701152090571 64] [IN]  PID: 11394  TASK: ffff880f9f971500  CPU: 4   COMMAND: "mdt02_047"
[6701152090531 77] [IN]  PID: 774    TASK: ffff88102f377540  CPU: 4   COMMAND: "mdt02_002"
[6701152090057 03] [IN]  PID: 11277  TASK: ffff880fa834b540  CPU: 14  COMMAND: "mdt03_023"
[6701152090036 67] [IN]  PID: 11301  TASK: ffff880fff02f540  CPU: 10  COMMAND: "mdt01_023"
[6701152090023 32] [IN]  PID: 11534  TASK: ffff880f9f718ae0  CPU: 5   COMMAND: "mdt02_081"
[6701152090019 43] [IN]  PID: 11344  TASK: ffff881026a27540  CPU: 3   COMMAND: "mdt01_034"
[6701152089994 23] [IN]  PID: 11255  TASK: ffff880ff491c040  CPU: 15  COMMAND: "mdt03_015"
[6701152089989 57] [IN]  PID: 4603   TASK: ffff880fd9674ae0  CPU: 2   COMMAND: "mdt01_006"
[6701152089988 05] [IN]  PID: 11366  TASK: ffff881026a40aa0  CPU: 10  COMMAND: "mdt01_039"
[6701152089939 85] [IN]  PID: 11300  TASK: ffff880fd6e8f500  CPU: 12  COMMAND: "mdt02_023"
[skipped 10]
[6701152089313 33] [IN]  PID: 11380  TASK: ffff880f9f83aae0  CPU: 6   COMMAND: "mdt03_046"
[6699645031368 81] [IN]  PID: 11499  TASK: ffff880f9f4d9540  CPU: 6   COMMAND: "mdt03_073"
[6699645021391 56] [IN]  PID: 11423  TASK: ffff880fc8ec8080  CPU: 15  COMMAND: "mdt03_057"
[6699644835372 74] [IN]  PID: 11365  TASK: ffff881026912080  CPU: 12  COMMAND: "mdt02_040"
[6699644835353 89] [IN]  PID: 11520  TASK: ffff880f9f6e9500  CPU: 12  COMMAND: "mdt02_078"
[6699644835335 17] [IN]  PID: 11528  TASK: ffff880f9f007540  CPU: 12  COMMAND: "mdt02_079"
[skipping 270]
[6699644741562 74] [IN]  PID: 11379  TASK: ffff880f9f83b540  CPU: 12  COMMAND: "mdt02_044"
```

crash – sample outputs - bt

backtraces

- Very helpful once processes identified (or to identify them)

```
 crash> bt 11454
PID: 11454  TASK: ffff880fcb367500  CPU: 11  COMMAND: "mdt01_062"
 #0 [ffff880fca83dc40] schedule at ffffffff81525cf0
 #1 [ffff880fca83dd08] ptlrpc_wait_event at ffffffffa08f5a15 [ptlrpc]
 #2 [ffff880fca83dda8] ptlrpc_main at ffffffffa08ff3cf [ptlrpc]
 #3 [ffff880fca83dee8] kthread at ffffffff8109ac66
 #4 [ffff880fca83df48] kernel_thread at ffffffff8100c20a

crash> bt 11501
PID: 11501  TASK: ffff880f9f4d8080  CPU: 4   COMMAND: "mdt02_074"
 #0 [ffff880f9f4e3850] schedule at ffffffff81525cf0
 #1 [ffff880f9f4e3918] ldlm_completion_ast at ffffffffa08c27b5 [ptlrpc]
 #2 [ffff880f9f4e39b8] ldlm_cli_enqueue_local at ffffffffa08c1b4e [ptlrpc]
 #3 [ffff880f9f4e3a38] mdt_object_lock0 at ffffffffa0e46c5c [mdt]
 #4 [ffff880f9f4e3ae8] mdt_object_lock at ffffffffa0e476a4 [mdt]
 #5 [ffff880f9f4e3af8] mdt_getattr_name_lock at ffffffffa0e5148d [mdt]
 #6 [ffff880f9f4e3b98] mdt_intent_getattr at ffffffffa0e52692 [mdt]
 #7 [ffff880f9f4e3be8] mdt_intent_policy at ffffffffa0e4164e [mdt]
 #8 [ffff880f9f4e3c58] ldlm_lock_enqueue at ffffffffa08a1869 [ptlrpc]
 #9 [ffff880f9f4e3cb8] ldlm_handle_enqueue0 at ffffffffa08ce1ab [ptlrpc]
#10 [ffff880f9f4e3d28] tgt_enqueue at ffffffffa094f5f1 [ptlrpc]
#11 [ffff880f9f4e3d48] tgt_request_handle at ffffffffa095007e [ptlrpc]
#12 [ffff880f9f4e3da8] ptlrpc_main at ffffffffa08ff8d9 [ptlrpc]
#13 [ffff880f9f4e3ee8] kthread at ffffffff8109ac66
#14 [ffff880f9f4e3f48] kernel_thread at ffffffff8100c20a
```

crash – sample outputs - bt

```
crash> foreach 'mdt0[0-9]_[0-9]*' bt > foreach_bt
$ awk '/PID:/ {PID=$2;PNAME=$8}
       /^ ?#/ {print PNAME": "$1" "$3" "$5}' foreach_bt | \
    clubak -c
---------------
11520,768,[...],11256,11488 (282)
---------------
 #0 schedule ffffffff81525cf0
 #1 ldlm_completion_ast ffffffffa08c27b5
 #2 ldlm_cli_enqueue_local ffffffffa08c1b4e
 #3 mdt_object_lock0 ffffffffa0e46c5c
 #4 mdt_object_lock ffffffffa0e476a4
 #5 mdt_getattr_name_lock ffffffffa0e5148d
 #6 mdt_intent_getattr ffffffffa0e52692
 #7 mdt_intent_policy ffffffffa0e4164e
 #8 ldlm_lock_enqueue ffffffffa08a1869
 #9 ldlm_handle_enqueue0 ffffffffa08ce1ab
 #10 tgt_enqueue ffffffffa094f5f1
 #11 tgt_request_handle ffffffffa095007e
 #12 ptlrpc_main ffffffffa08ff8d9
 #13 kthread ffffffff8109ac66
 #14 kernel_thread ffffffff8100c20a
---------------
769,11390,[...],11521 (36)
---------------
 #0 schedule ffffffff81525cf0
 #1 ldlm_completion_ast ffffffffa08c27b5
 #2 ldlm_cli_enqueue_local ffffffffa08c1b4e
 #3 mdt_object_lock0 ffffffffa0e46e64
 #4 mdt_object_lock ffffffffa0e476a4
 #5 mdt_object_find_lock ffffffffa0e479d1
 #6 mdt_reint_open ffffffffa0e71eae
 #7 mdt_reint_rec ffffffffa0e5c8ad
 #8 mdt_reint_internal ffffffffa0e42acb
 #9 mdt_intent_reint ffffffffa0e42f56
 #10 mdt_intent_policy ffffffffa0e4164e
 #11 ldlm_lock_enqueue ffffffffa08a1869
 #12 ldlm_handle_enqueue0 ffffffffa08ce1ab
 #13 tgt_enqueue ffffffffa094f5f1
 #14 tgt_request_handle ffffffffa095007e
 #15 ptlrpc_main ffffffffa08ff8d9
 #16 kthread ffffffff8109ac66
 #17 kernel_thread ffffffff8100c20a
```

```
---------------
775,11260,771 (3)
---------------
 #0 schedule ffffffff81525cf0
 #1 schedule_timeout ffffffff81526b52
 #2 ldlm_completion_ast ffffffffa08c2721
 #3 ldlm_cli_enqueue_local ffffffffa08c1b4e
 #4 mdt_object_lock0 ffffffffa0e46c5c
 #5 mdt_object_lock ffffffffa0e476a4
 #6 mdt_getattr_name_lock ffffffffa0e5148d
 #7 mdt_intent_getattr ffffffffa0e52692
 #8 mdt_intent_policy ffffffffa0e4164e
 #9 ldlm_lock_enqueue ffffffffa08a1869
 #10 ldlm_handle_enqueue0 ffffffffa08ce1ab
 #11 tgt_enqueue ffffffffa094f5f1
 #12 tgt_request_handle ffffffffa095007e
 #13 ptlrpc_main ffffffffa08ff8d9
 #14 kthread ffffffff8109ac66
 #15 kernel_thread ffffffff8100c20a
---------------
11247,4596,4636 (3)
---------------
 #0 schedule ffffffff81525cf0
 #1 schedule_timeout ffffffff81526b52
 #2 ldlm_completion_ast ffffffffa08c2721
 #3 ldlm_cli_enqueue_local ffffffffa08c1b4e
 #4 mdt_object_lock0 ffffffffa0e46e64
 #5 mdt_object_lock ffffffffa0e476a4
 #6 mdt_object_find_lock ffffffffa0e479d1
 #7 mdt_reint_open ffffffffa0e71eae
 #8 mdt_reint_rec ffffffffa0e5c8ad
 #9 mdt_reint_internal ffffffffa0e42acb
 #10 mdt_intent_reint ffffffffa0e42f56
 #11 mdt_intent_policy ffffffffa0e4164e
 #12 ldlm_lock_enqueue ffffffffa08a1869
 #13 ldlm_handle_enqueue0 ffffffffa08ce1ab
 #14 tgt_enqueue ffffffffa094f5f1
 #15 tgt_request_handle ffffffffa095007e
 #16 ptlrpc_main ffffffffa08ff8d9
 #17 kthread ffffffff8109ac66
 #18 kernel_thread ffffffff8100c20a
---------------
11466,11342,11281,[...],11454,11484,11482 (12)
---------------
 #0 schedule ffffffff81525cf0
 #1 ptlrpc_wait_event ffffffffa08f5a15
 #2 ptlrpc_main ffffffffa08ff3cf
 #3 kthread ffffffff8109ac66
 #4 kernel_thread ffffffff8100c20a
```

crash – sample outputs – fid/nid

Find file/client

- Normally use 'disass' to look at assembler and find variables on the stack
- But sometimes we can cheat! (thank you, SLAB)

```
crash> bt -FF 11256
…
    ffff880fd4f33c20: tgt_dlm_handlers [ffff880fb3140080:ldlm_locks]
    ffff880fd4f33c30: ffff880fd4f33ce8 ffff880fd4f33ce0
    ffff880fd4f33c40: [ffff881026cd6200:ldlm_resources] [ffff881071ca1800:size-512]
    ffff880fd4f33c50: ffff880fd4f33cb0 ldlm_lock_enqueue+297
 #8 [ffff880fd4f33c58] ldlm_lock_enqueue at ffffffffa08a1869 [ptlrpc]
    ffff880fd4f33c60: [ffff880fb3140001:ldlm_locks] 00000000d4f33c80
…
    ffff880fd4f33c70: ffff880fd4f33cb0 00000000a05473fd
    ffff880fd4f33c80: [ffff880fdd838400:size-512] ffffc90040cef728
    ffff880fd4f33c90: [ffff880fca927980:ptlrpc_cache] 0000000000000000
    ffff880fd4f33ca0: [ffff880fca927980:ptlrpc_cache] [ffff881071ca1800:size-512]
    ffff880fd4f33cb0: ffff880fd4f33d20 ldlm_handle_enqueue0+1451
 #9 [ffff880fd4f33cb8] ldlm_handle_enqueue0 at ffffffffa08ce1ab [ptlrpc]
    ffff880fd4f33cc0: ffff881000000000 ffff880f00000000
…
```

Get associated file

```
crash> mod -s mdt
crash> set radix 16
output radix: 16 (hex)
crash> struct ldlm_resource.lr_name ffff881026cd6200
  lr_name = {
    name = {0x200006031, 0xc3b4, 0x0, 0x0}
  }
$ lfs fid2path /path/to/fsroot 0x200006031:0xc3b4
/path/to/fsroot/sorry/cant/give/real/paths
```

Get request's peer nid

```
crash> mod -s ptlrpc
crash> struct ptlrpc_request.rq_peer.nid ffff880fca927980
  rq_peer.nid = 0x500010a020304,
crash> net -N 0x500010a020304
4.3.2.10
# 10.2.3.4@o2ib1
```

crash – sample outputs – fid/nid

Abusing kmem_cache names

- Just a peek at how to figure proper typing

```
$ cat $lustresrc/lustre/ldlm/ldlm_lockd.c
…
    ldlm_resource_slab = kmem_cache_create("ldlm_resources",
                           sizeof(struct ldlm_resource), 0,
                           SLAB_HWCACHE_ALIGN, NULL);
…
$ cat $lustresrc/lustre/ldlm/ldlm_resource.c
…
static struct ldlm_resource *ldlm_resource_new(void)
{
    struct ldlm_resource *res;
    int idx;

    OBD_SLAB_ALLOC_PTR_GFP(res, ldlm_resource_slab, GFP_NOFS);
…
```

- Also warning: the address isn't necessarily the start of the structure
  - Will match anything *within* slab-allocated area
  - Check what the address exactly as usual
    - declarations at start of function & arguments
  - crash struct helper has options that behave like container_of
    - help struct → *-l offset* (accepts structure.member syntax)

crash – sample outputs – fid/nid

Industrialisation process & other “nice” structures

- Can use foreach once position on stack identified

```
    ffff880fe12a3ca0: [ffff880fc82700c0:ptlrpc_cache] [ffff881071ca1800:size-512]
    ffff880fe12a3cb0: ffff880fe12a3d20 ldlm_handle_enqueue0+0x5ab
…
crash> foreach 'mdt0[0-9]_[0-9]*' bt -FF -R ldlm_handle_enqueue0 |
       grep -B1 'ldlm_handle_enqueue0+0x5ab' |
       grep -oE 'ffff[0-9a-f]*:ptlrpc_cache' |
       cut -d: -f1 > ptlrpc_requests
# requests contains addresses e.g. ffff880fc82700c0

crash> struct ptlrpc_request.rq_peer.nid < ptlrpc_requests |
       grep -oE '0x[0-9a-f]*' > nids
# nids contains.. nids e.g. 0x5000c0a020304

crash> net -N < nids | sort | uniq -c | sort -n
# ...
#     1 42.12.2.10
#     1 72.15.2.10
# ...
#    10 12.17.2.10
#    10 13.17.2.10
#    10 14.17.2.10
```

- Can help identify a job or clients to forcefully evict

Further reading...

Crash can do more!

- More useful structures to look for
  - *struct structure* with no pointer describes structure

```
struct ldlm_resource
 lr_name
 lr_granted, lr_waiting

struct ldlm_lock
 l_resource
 l_last_activity, l_last_used
 l_pid, l_readers, l_writers
```

```
struct mdt_thread_info (no slab alloc)
 struct mdt_reint_record mti_rr
 ldlm_policy_data_t mti_policy
 struct lu_fid mti_tmp_fid1/mti_tmp_fid2

Which fields are used here depend on function
stack, don't go guessing things through fids that
might be leftovers from previous RPC!
```

- More commands
  - kmem - informations about memory, slabs, etc
  - list - list traversal
  - help!

DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Tools: gdb

VM-specific debugging

- Start qemu's gdb-server & hook in!
- Helps to have kernel's gdb scripts
  - Not compiled in by default
  - Can use anyway, but need to be aware of versions (depends on kernel structures to load modules, etc - pick from *script/gdb* in sources since v4.0)
- Lets you add breakpoints anywhere in the kernel
  - With *cond*, *commands*, *watchpoint*, etc
- Running context also means
  - Easier access to variables (*info local*)
  - *next*, *step*, *finish*
    - although anything that can *schedule()* will likely get you lost

gdb example

Let's try

```
[martinet@cobalt172 ~]$ pcocc monitor-cmd vm0 gdbserver tcp::1234
[martinet@cobalt172 ~]$ ssh cobalt1822
[martinet@cobalt1822 ~]$ cd $CCCSCRATCHDIR/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64
[martinet@cobalt1822 2.6.32-573.18.1.ocean1.el6.x86_64]$ gdb vmlinux
(gdb) set pagination off
(gdb) source scripts/gdb/vmlinux-gdb.py
(gdb) set substitute-path /usr/src/debug /path/to/scratchdir/debuginfo-ocean1/usr/src/debug
(gdb) target remote localhost:1234
(gdb) lx-symbols
loading vmlinux
scanning for modules in /path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64
loading @0xffffffffa0e49000:
/path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64/extra/kernel/fs/lustre/osp.ko.debug
loading @0xffffffffa0df1000:
/path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64/extra/kernel/fs/lustre/mdd.ko.debug
loading @0xffffffffa0dd0000:
/path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64/extra/kernel/fs/lustre/lfsck.ko.debug
loading @0xffffffffa0d7f000:
/path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64/extra/kernel/fs/lustre/lod.ko.debug
loading @0xffffffffa0c9f000:
/path/to/scratchdir/debuginfo-ocean1/usr/lib/debug/lib/modules/2.6.32-573.18.1.ocean1.el6.x86_64/extra/kernel/fs/lustre/mdt.ko.debug
…
(gdb) b ldlm_lock_cancel
Breakpoint 1 at 0xffffffffa05d3fdc: file /usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lock.c, line 2154.
(gdb) commands
Type commands for breakpoint(s) 1, one per line.
End with a line saying just "end".
>bt
>cont
>end
(gdb) cont
Continuing.
```

gdb example

```
Breakpoint 1, ldlm_lock_cancel (lock=0xffff88074fa0e7c0) at /usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lock.c:2154
2154                    ENTRY;
#0  ldlm_lock_cancel (lock=0xffff88074fa0e7c0) at /usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lock.c:2154
#1  0xffffffffa05edeca in ldlm_cli_cancel_local (lock=0xffff88074fa0e7c0) at
/usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_request.c:1154
#2  0xffffffffa05f2730 in ldlm_cli_cancel (lockh=<optimized out>, cancel_flags=LCF_ASYNC) at
/usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_request.c:1366
#3  0xffffffffa05f2ac7 in ldlm_blocking_ast_nocheck (lock=0xffff88074fa0e7c0) at
/usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_request.c:330
#4  0xffffffffa0ca3f70 in mdt_blocking_ast (lock=0xffff88074fa0e7c0, desc=<optimized out>, data=<optimized out>, flag=<optimized out>)
at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:2621
#5  0xffffffffa05f6800 in ldlm_handle_bl_callback (ns=<optimized out>, ld=0x0 <per_cpu__irq_stack_union>, lock=0xffff88074fa0e7c0)
at /usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lockd.c:1725
#6  0xffffffffa05d73ee in ldlm_lock_decref_internal (lock=0xffff88074fa0e7c0, mode=<optimized out>) at
/usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lock.c:910
#7  0xffffffffa05d85f9 in ldlm_lock_decref (lockh=0xffff88072d23e028, mode=8) at
/usr/src/debug/lustre-2.5.5/lustre/ldlm/ldlm_lock.c:947
#8  0xffffffffa0ca37b3 in mdt_fid_unlock (mode=LCK_CW, lh=0xffff88072d23e028) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_internal.h:1120
#9  mdt_save_lock (info=0xffff88072d23e000, h=0xffff88072d23e028, mode=LCK_CW, decref=<optimized out>) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:2869
#10 0xffffffffa0ca3aba in mdt_object_unlock (info=0xffff88072d23e000, o=<optimized out>, lh=0xffff88072d23e010, decref=0) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:2924
#11 0xffffffffa0ca56a7 in mdt_object_unlock_put (info=0xffff88072d23e000, o=0xffff8807507882b0, lh=<optimized out>, decref=<optimized
out>) at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:2959
#12 0xffffffffa0cc5ea9 in mdt_md_create (info=0xffff88072d23e000) at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_reint.c:387
#13 mdt_reint_create (info=0xffff88072d23e000, lhc=<optimized out>) at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_reint.c:669
#14 0xffffffffa0cc3641 in mdt_reint_rec (info=0xffff88072d23e000, lhc=0x0 <per_cpu__irq_stack_union>) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_reint.c:1513
#15 0xffffffffa0cad003 in mdt_reint_internal (info=0xffff88072d23e000, lhc=0x0 <per_cpu__irq_stack_union>, op=2) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:1917
#16 0xffffffffa0cad304 in mdt_reint (info=0xffff88072d23e000) at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:1978
#17 0xffffffffa0cab11a in mdt_req_handle (req=0xffff880738d71c00, h=0xffffffffa0d1fbd8 <mdt_mds_ops+120>, info=0xffff88072d23e000)
at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:3194
#18 mdt_handle0 (supported=0xffffffff00000008, info=0xffff88072d23e000, req=0xffff880738d71c00) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:3577
#19 mdt_handle_common (req=0xffff880738d71c00, supported=0xffffffff00000008) at
/usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_handler.c:3625
#20 0xffffffffa0ce86c5 in mds_regular_handle (req=<optimized out>) at /usr/src/debug/lustre-2.5.5/lustre/mdt/mdt_mds.c:275
#21 0xffffffffa062e0f5 in ptlrpc_server_handle_request ()
#22 0xffffffffa063087d in ptlrpc_main ()
#23 0xffffffff810a0fce in kthread (_create=0xffff88074efb0940) at kernel/kthread.c:88
#24 0xffffffff8100c28a in child_rip ()
#25 0x0000000000000000 in ?? ()
```

Gotchas

- Break for too long doesn't mean timers are stopped
  - easy to see rare timeouts
  - like any multithreaded application, 'next' can race you in the background
- Cannot open crash dumps,
- not as powerful as crash to explore processes/resources
  - Although “easy” to script in python

```
import gdb
from linux import utils
list_head = utils.CachedType("struct list_head")

def list_check(head):
    nb = 0
    if (head.type == list_head.get_type().pointer()):
        head = head.dereference()
    elif (head.type != list_head.get_type()):
        raise gdb.GdbError('argument must be of type
                           (struct list_head [*])')
    c = head
    try:
        gdb.write("Starting with: {}\n".format(c))
    except gdb.MemoryError:
        gdb.write('head is not accessible\n')
        return
```

head of linux/scripts/gdb/linux/lists.py

```
    while True:
        p = c['prev'].dereference()
        n = c['next'].dereference()
        try:
            if p['next'] != c.address:
                gdb.write('prev.next != current: '
                          'current@{current_addr}={current} '
                          'prev@{p_addr}={p}\n'.format(
                              current_addr=c.address,
                              current=c,
                              p_addr=p.address,
                              p=p,
                          ))
                return
        except gdb.MemoryError:
…
```

DE LA RECHERCHE À L'INDUSTRIE

Tools: systemtap

www.cea.fr

Live system instrumentation and workarounds

- Compile a small kernel module that can hook in any
  - function's entry, return point
  - (most) line numbers...
- Log values
  - arguments (*stap -L "probe point"* to list available variables)
    - Complete structure introspection/logging ($struct$$)
  - informations derived from arguments by calling other functions
    - Can do anything with embedded C code!
  - function execution time statistics/call graphs
- Change return value, local variables
- Security hotfixes/bug workarounds
  - Cannot force early return, but often can modify arguments to cause EINVAL

Example – local workaround + log

LU-5642: getxattr failing with EIO

- Sometimes getxattr fails with EIO
- Simple workaround until fixed upstream

```
#!/usr/bin/stap -g

global EIO = -5
global EAGAIN = -11

probe begin {
    print("LU-5642 stap started. Press ^C to exit\n")
}

function syslog(msg: string) %{
    printk("stap lu_5642: %s\n", STAP_ARG_msg);
%}

probe module("lustre").function("ll_xattr_cache_refill").return {
    if ($return == EIO && uid() > 1000) {
        slurm_jobid = env_var("SLURM_JOBID")
        syslog(sprintf("JOBID %s: %s(%d) got EIO, changing to EAGAIN (inode %p)",
                       slurm_jobid, execname(), pid(), $inode))
        $return = EAGAIN
    }
}
```

Example – security hotfix – probe usage

Probe usage

- Manual run

```
# yum install systemtap-devel kernel-headers
# stap -g -v lu_5642.stp
Pass 1: parsed user script and 113 library script(s) using 220336virt/37716res/3048shr/35148data kb, in 200usr/20sys/212real ms.
Pass 2: analyzed script: 4 probe(s), 32 function(s), 3 embed(s), 0 global(s) using 319160virt/137688res/4208shr/133972data kb, in
1790usr/700sys/2448real ms.
Pass 3: translated to C into "/tmp/stap7H7tB3/stap_46c9da9de15aa2342b70f3ccee7b96d8_20032_src.c" using
319160virt/137992res/4512shr/133972data kb, in 70usr/70sys/139real ms.
Pass 4: compiled C into "stap_46c9da9de15aa2342b70f3ccee7b96d8_20032.ko" in 8550usr/1540sys/10158real ms.
Pass 5: starting run.
LU-5642 stap started. Press ^C to exit
```

- Production coverage
  - only need systemtap-runtime package
  - copy .ko to /lib/modules/$(uname -r)/systemtap
  - staprun <modulename> in a service at boot

- stap -L

```
# stap -L 'module("lustre").function("ll_xattr_cache_refill")'
module("lustre").function("ll_xattr_cache_refill@/usr/src/debug/lustre-2.7.1-d54e7ef/lustre/llite/xattr_cache.c:384")
$inode:struct inode* $oit:struct lookup_intent* $__func__:char[] const
```

DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

Some closure on the problem

Some more analysis

Our actual problem

- We notice a pattern such that:
  - Many clients hold a PR lock on directory
  - Some clients requests a CW lock, one gets elected for upgrade
    - MDT cancels all PR locks held
    - CW lock is NOT granted quickly: some cancels are slow/not happening?
    - Clients who did cancel send in new PR lock requests (waiting)
  - New PR lock requests starve all threads and MDT is just waiting for first clients to cancel until timeout happens
  - Eventually recovers through timeouts after a veeery long while (many writers)
- Work-around kind of happened with a system update (selinux-policy disabling security.selinux xattr on lustre)
  - Thought it was fixed with 2.7.2 until Friday evening... Back on it! :)

Conclusion

Going further...

- The bug happens as well with a single file in directory, even if the file already exists
  - Might be worth checking if entry exists with read lock on directory before upgrading lock?
- There actually are a few free threads, unsure why behavior is full lock-down

Looking back!

- Annoying bug, but being able to reproduce complex problems (hundred of clients!) is cool
  - Without impacting production
  - Easier to export logs (black site...)
  - Useful for simpler problems too!

Thank you for your attention !

Questions ?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA / DAM Ile-de-France | Bruyères-le-Châtel - 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00

DAM Île-de-France

Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019

pcocc configuration

Template file: ~/.pcocc/templates.yaml

```
scs5:
  image: '%{env:CCCSCRATCHDIR}/pcocc/images/scs5'
  resource-set: cluster-ib
  description: Bull compute node
  user-data:  '%{env:CCCSCRATCHDIR}/pcocc/userdata/scs5'

ocean:
  image: '%{env:CCCSCRATCHDIR}/pcocc/images/ocean1.2'
  resource-set: cluster-ib
  description: Lustre server
  user-data:  '%{env:CCCSCRATCHDIR}/pcocc/userdata/ocean1'

mds0:
  inherits: ocean
  persistent-drives:
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/mgt'
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/mdt0'

oss0:
  inherits: ocean
  persistent-drives:
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/ost0'
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/ost1'

oss1:
  inherits: ocean
  persistent-drives:
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/ost2'
    - '%{env:CCCSCRATCHDIR}/pcocc/images/lustre/ost3'
```

hang reproducer – MPI version

```
int main(int argc, char** argv) {
    int rc, fd, i;

    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    if (world_rank == 0) {
        rc = mkdir("/ccc/testfs/userdir/foo.d", 0755);
        printf("mkdir rc/errno: %d/%d\n", rc, errno);
    }

    MPI_Barrier(MPI_COMM_WORLD);

    if (world_rank % 3 == 0) {
        char filename[1000];
        snprintf(filename, 1000, "/ccc/testfs/userdir/foo.d/filerank.%d", world_rank);
        for (i=0; i<100; i++) {
            fd = open(filename, O_RDWR|O_CREAT, 0644);

            if (fd < 0) {
                printf("Proc %d failed open with errno %d\n", world_rank, errno);
                break;
            }
            close(fd);
        }
    } else {
        for (i=0; i<4096; i++) {
            struct stat buf;

            rc = stat("/ccc/testfs/userdir/foo.d", &buf);

            if (rc < 0) {
                printf("Proc %d failed stat with errno %d\n", world_rank, errno);
                break;
            }
        }
    }

    MPI_Finalize();

    return 0;
}
```

Example – local workaround + log

LU-6471: Unexpected Lustre Client LBUG in llog_write()

- Got some LBUG

```
<3>[ 2614.548692] LustreError: 15044:0:(llog_cat.c:164:llog_cat_id2handle())
work2-MDT0000-mdc-ffff882050fe9400: error opening log id 0x11bf:1:0: rc = -2
<3>[ 2614.562046] LustreError: 15044:0:(llog_cat.c:536:llog_cat_process_cb())
work2-MDT0000-mdc-ffff882050fe9400: cannot find handle for llog 0x11bf:1: -2
<0>[ 2614.575475] LustreError: 15044:0:(llog.c:850:llog_write()) ASSERTION(
loghandle->lgh_obj != ((void *)0) ) failed:
<0>[ 2614.585959] LustreError: 15044:0:(llog.c:850:llog_write()) LBUG
```

- Simple workaround until fixed upstream

```
#!/usr/bin/env stap

probe module("obdclass").statement(
"llog_cat_process_cb@/usr/src/debug/lustre-2.5.3.90/lustre/obdclass/llog_cat.c:537"
)
{
  if ( $rc == -2 || $rc == -116 ) {
    if ( $cat_llh->lgh_obj == NULL ) {
      printf("llog_cat_process_cb:537 : RC = %d cat_llh->lgh_obj null !\n", $rc);
      $rc = 0;
    }
  }
}
```

Example – security hotfix - CVE-2016-4565

kernel/IB: Restrict use of the write() interface

- Public arbitrary write exploit available
  - discussions on linux-rdma@vger.kernel.org, oss-security@lists.openwall.com
  - see mail archives https://marc.info/?l=oss-security&m=146259498215687&w=2

- Very simple to use

```
bash-4.2$ gcc -o CVE-2016-4565 CVE-2016-4565.c
bash-4.2$ grep vm_swappiness /proc/kallsyms
ffffffff819bfe70 D vm_swappiness
bash-4.2$ cat /proc/sys/vm/swappiness
30
bash-4.2$ ./CVE-2016-4565 0xffffffff819bfe70
that probably worked? clobber_kaddr(0xffffffff819bfe70)=32
bash-4.2$ cat /proc/sys/vm/swappiness
0
```

- Cannot exactly wait for update
  - Took vendor 2 months to release a fix (May 6 - July 10)
  - Shipped for (part of) production end of August... Still not installed everywhere!

Example – security hotfix – workaround probe

```
probe begin {
                print ("CVE-2016-4565 stap started. Type Ctrl-C to exit\n")
}

function syslog(msg : string) %{
        printk("stap CVE-2016-4565: %s\n", STAP_ARG_msg);
%}

function task_fullpath(task : long) {
        task_fs = @cast(task, "task_struct", "kernel<linux/sched.h>")->fs
        t_dentry = @cast(task_fs, "fs_struct", "kernel")->pwd->dentry
        vfsmnt = @cast(task_fs, "fs_struct", "kernel")->pwd->mnt
        return task_dentry_path(task, t_dentry, vfsmnt)
}

function ib_safe_file_access:long (filp: long) %{
        struct file *filp = (struct file*)STAP_ARG_filp;

        STAP_RETURN(filp->f_cred == current_cred() && segment_eq(get_fs(), USER_DS));
%}


probe module("ib_ucm").function("ib_ucm_write") {
        if (!ib_safe_file_access($filp)) {
                syslog(sprintf("%s(%d), path %s, user %u\n", execname(), pid(), task_fullpath(task_current()), uid()))
                $len = 0
       }
}

probe module("rdma_ucm").function("ucma_write") {
        if (!ib_safe_file_access($filp)) {
                syslog(sprintf("%s(%d), path %s, user %u\n", execname(), pid(), task_fullpath(task_current()), uid()))
                $len = 0
       }
}

probe module("ib_uverbs").function("ib_uverbs_write") {
        if (!ib_safe_file_access($filp)) {
                syslog(sprintf("%s(%d), path %s, user %u\n", execname(), pid(), task_fullpath(task_current()), uid()))
                $count = 0
       }
}
```