Tips and Tricks for Diagnosing Lustre Problems on Cray Systems

Cory Spitz and Ann Koehler

Cray Inc.

5/25/2011

CRAY
THE SUPERCOMPUTER COMPANY

Introduction

- Lustre is a critical system resources
- Therefore, problems need to be quickly diagnosed
- Administrators and operators need to collect the necessary debug data the first time a problem arises
  - Can’t interrupt production workload to investigate problems
  - Can’t depend on Cray to reproduce
- Performance is important too
  - Cray systems are supercomputers after all
- The paper and this talk cover a broad range of topics
  - The paper covers much more

Agenda

- Working with console logs and the *syslog*
- How to read console logs
- What to look for
- Common failures
- Collecting additional debug data
- Performance

Working with console logs and the *syslog*

- Console logs are the go-to resource for Lustre problems
- But, Lustre logs can be overwhelming, especially at scale
  - Lustre is chatty ;)
  - Sequential messages from a single node can span pages
  - `printk` rate limiting can only go so far
- Use `lustrelogs.sh` to separate server console logs
  - Writes out per-server logs with unambiguous names
    - Even if failover occurred
  - E.g. `oss1.c1-0c0s0n3.nid00131.OST0001`
  - Appendix A of paper
- Minor `printk` level messages go to the *syslog*

How to read console logs

- Understand node references
  - Lustre identifies endpoints based on their LNET names
  - `<#>@<lnet>`, e.g. `100@gni` identifies `nid00100`
  - Console logs are prefixed with Cray *cname*
  - Cross-reference names via `xtprocadmin` or `/etc/hosts`
  - For esFS, the IPoIB address identifies an external node
    - But, IPoIB not used for transport
- Understand error codes
  - Lustre uses standard Linux/POSIX *errno* values
  - E.g. "`the ost_write operation failed with -30`"
  - Keep `errno-base.h` and `errno.h` from `/usr/include/asm-generic` handy

What to look for in console logs

- Identify major (node) faults
  - `LBUG`
    - Cray enables `panic_on_lbug` by default
  - `ASSERT`
  - `Oops`
  - `Call Trace`
    - Not necessarily fatal
    - Stack trace emitted
- Also look for file client specific problems
  - `evict`
  - `suspect`
  - `admindown`
- Ensure sane configuration and proper startup
  - `` `lustre_control.sh <fsname>.fs_defs verify_config` ``
  - `boot:/tmp/lustre_control.<id>`

Client eviction

- Eviction results in lost data but clients can stay ‘up’
  - Can even pass NHC file system test
  - Users may characterize this as corruption
- Common causes
  - Client fails to ping server within 1.5x `obd_timeout`
  - Client fails to handle blocking lock callback within `ldlm_timeout`
  - Failed or flaky router or routes
    - Although clients resend RPCs
    - Servers do not resend lock callbacks

Client eviction examples

- Client side:

```
LustreError: 11-0: an error occurred while communicating
with 135@ptl. The ldlm_enqueue operation failed with -107
LustreError: 167-0: This client was evicted by test-MDT0000;
in progress operations using this service will fail.
```

- Server side:

```
Lustre: MGS: haven't heard from client 73c68998-6ada-5df5-
fa9a-9cbbe5c46866 (at 7@ptl) in 679 seconds. I think it's
dead, and I am evicting it.
```

- Or:

```
LustreError: 0:0:(ldlm_lockd.c:305:waiting_locks_callback())
### lock callback timer expired after 603s: evicting client
at 415@ptl  ns: mds-test-MDT0000_UUID lock:
ffff88007018b800/0x649105220915890 6 lrc: 3/0,0 mode: CR/CR
res: 4348859/3527105419 bits 0x3 rrc: 5 type: IBT flags:
0x4000020 remote: 0x6ca282feb4c7392 expref: 13 pid: 11168
timeout: 4296831002
```

Client eviction examples

- Client side:

```
LustreError: 167-0: This client was evicted by lustrefs-
OST0002; in progress operations using this service will
fail.
```

  Server side:

```
LustreError: 138-a: lustrefs-OST0002: A client on nid
171@gni was evicted due to a lock blocking callback to
171@gni timed out: rc -4
```

- And:

```
LustreError: 0:0:(ldlm_lockd.c:305:waiting_locks_callback())
### lock callback timer expired after 105s: evicting client
at 171@gni  ns: filter-lustrefs-OST0002_UUID lock:
ffff8803c11a8000/0x69ba7544a5270d3d lrc: 4/0,0 mode: PR/PR
res: 136687655/0 rrc: 3 type: EXT [0->18446744073709551615]
(req 0->4095) flags: 0x10020 remote: 0x59d12fa603479bf2
expref: 21 pid: 8567 timeout 4299954934
```

Gemini HW errors and resiliency features

- "Stack reset" upon critical HW errors
  - Gather critical errors via `xthwerrlog -c crit -f <file>`
  - NIC is reset
  - gnilnd pauses all transfers and re-establishes connections
  - Mechanism to ensure no lagging RDMA
    - `n_mdd_held` field in `/proc/kgnilnd/stats`
  - *errno* -131, `ENOTRECOVERABLE` for gnilnd, but Lustre can recover
- Quiesce and reroute for failed links
  - `at_min` and `ldlm_timeout` tuned up to 70s
- Appendix C in paper describe gnilnd codes and meanings

```
LNet: critical hardware error: resetting all resources (count 1)
LNet:3980:0:(gnilnd.c:645:kgnilnd_complete_closed_conn()) Closed
conn 0xffff880614068800->0@gni (errno -131): canceled 1 TX, 0/0
RDMA
LNet: critical hardware error: All threads awake!
LNet: successful reset of all hardware resources
```

Collecting debug kernel traces (dk log)

- Lustre Operations Manual Chapter 24, Lustre debugging
- Turn on full debug:
  - `` `lctl set_param debug=-1` ``
- Increase the size of the ring buffer:
  - `` `lctl set_param debug_mb=400` ``
- Start fresh:
  - `` `lctl clear` ``
- Annotate:
  - `` `lctl mark <annotation>` ``
- Collect (“1” not a typo, fast binary mode):
  - `` `lctl dk <file> 1` ``
- Or, enable `dump_on_timeout` or `dump_on_eviction`
- Convert dk log to human readable format and time scale
  - `sort_lctl.sh` and `lctl_daytime.sh` in Appendix B of paper

Additional debug data

- There is a wealth of data in Lustre `/proc` interfaces
- Most everything is documented in the Lustre Ops Manual
- Watch the clients “import” files
  - Shows connection status, rpc state counts, service estimates
  - `` `lctl get_param *.*.import` ``
  - Example on next slide
- Cleint and server side “stats”
  - `` `lctl get_param *.*.stats` `` or `` `llstat` ``
  - Shows counts, min and max time in µsecs, sum and sum squared
- Recovery status on servers
  - `` `lctl get_param *.*.recovery_status` ``
- LMT or `llobdstat` for real-time monitoring

Client *import* file

```
import:
    name: lustrefs-OST0001-osc-ffff8803fd227400
    target: lustrefs-OST0001_UUID
    state: FULL
    connect_flags: [write_grant, server_lock, version,
      request_portal, truncate_lock,
      max_byte_per_rpc, early_lock_cancel,
      adaptive_timeouts, lru_resize,
      alt_checksum_algorithm, version_recovery]
    import_flags: [replayable, pingable]
    connection:
       failover_nids: [26@gni, 137@gni]
       current_connection: 26@gni
       connection_attempts: 1
       generation: 1
       in-progress_invalidations: 0
[…]
```

```
rpcs:
       inflight: 0
       unregistering: 0
       timeouts: 0
       avg_waittime: 24121 usec
    service_estimates:
       services: 70 sec
       network: 70 sec
transactions:
       last_replay: 0
       peer_committed: 403726926456
       last_checked: 403726926456
    read_data_averages:
       bytes_per_rpc: 1028364
       usec_per_rpc: 41661
       MB_per_sec: 24.68
    write_data_averages:
       bytes_per_rpc: 1044982
       usec_per_rpc: 21721
       MB_per_sec: 48.10
```

Metadata performance

- Metadata performance is one of Lustre's biggest complaints
  - Usually voiced as the result of interactive usage
- Clients are limited to a single modifying metadata operation
- Only way to get more ops in flight is to add more nodes
- `max_rpcs_in_flight` parameter is for non-modifying ops
  - Tune up on interactive login nodes
- Users tend to make it worse
  - `` `ls -l` `` is expensive on Lustre
    - Be careful, ls is aliased to `` `ls -color=tty` ``
  - Really, it is `stat()` that is expensive
- Use `` `lfs check servers` `` instead of `` `/bin/df` ``

Bulk read/write performance

- Client side: `` `lctl get_param osc.*.rpc_stats` ``
- Server side: `` `lctl get_param obdfilter.*.brw_stats` ``
  - I/O times are reported
  - Looking for 1 MiB writes all the way through to disk
    - Avoid read-modify-write in HW RAID controller
    - And/or avoid cache mirroring depending on RAID type
  - Use *sd_iostats* data to see the effect of fs metadata (e.g. journals)
  - Unoptimal I/O is not an error
    - Could be silent errors (sector remapping, etc.)
    - Could be RAID rebuild
- Per client stats: `` `lctl get_param obdfilter.*.exports.*.brw_stats` ``
- OSS Read Cache
  - O_DIRECT cache semantics
  - `` `lctl set_param obdfilter.*.readcache_max_filesize=32M` ``

LNET performance

- Credits are key
  - Network Interface (NI) transmit (tx) credits
    - Maximum number of concurrent sends for the LNET
  - Peer tx credits
    - Number of concurrent sends to a single peer
- Credits are like semaphores
- NI and tx credits must be acquired to send to a remote peer
- If a credit isn't available the send is queued
- Monitor credit use
  - `/proc/sys/lnet/nis`
  - `/proc/sys/lnet/peers`
  - Negative numbers indicate queued sends
  - "min" column shows low water mark
  - If "min" is negative for 'normal' operation, consider tuning credits

LNET router performance

- Two more credits need to be acquired for router tx
- Router buffer credits
  - Router buffers hold bulk data for network bridging (RDMA)
    - Less than a page `tiny_router_buffers`
    - Page sized `small_router_buffers`
    - 1MiB `large_router_buffers`
- Peer router buffer credits
  - Number of router buffers used for a single peer
  - Defaults to LND peer tx credit count
    - Consider tuning lnet.ko module parameter `peer_buffer_credits`
- Monitor router credits
  - `/proc/sys/lnet/buffers`
  - `/proc/sys/lnet/peers`
  - Again, negative numbers indicate queued operation

Questions?