Scaling one single Lustre Filesystem up to 20K clients

September 28th, 2022

Aurélien CEDEYN / Gaël DELBARY
AGENDA

- Site update
- Context case study
- Credits and peer_credits
- CPU Partition Table
- High priority RPC nightmare
Site update
Compute centers at CEA/DAM

2 production compute centers:
- EXA: Defense application
- TGCC: European research
  - Hosting France Génomique (storage of DNA sequencing data)
  - Hosting CCRT (for industrial companies)
  - Hosting Human Brain project

1 lab compute center:
- R&D compute nodes
- R&D storage cluster

Compute power:
- EXA: 40 Pflops (~20K clients)
- TGCC: 25 Pflops (~6K clients)
- LAB: 1 Pflops (~200 clients)

2 production compute centers with a similar design:
- Nearly the same architecture, technologies, tools and system software
User interface: Lustre/HSM (Lustre 2.12.9++)
- Lustre 2.15 upgrade planned for Q2/Q3 2023
## CEA HPC storage numbers

<table>
<thead>
<tr>
<th>Production systems</th>
<th>Capacity</th>
<th>Throughput</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGCC</td>
<td>40 PB (total: 80 PB, HSM backend)</td>
<td>900 GB/s</td>
<td>5000 K</td>
</tr>
<tr>
<td>TERA</td>
<td>60 PB (total: 120 PB, HSM backend)</td>
<td>2.5 TB/s</td>
<td>25000 K</td>
</tr>
</tbody>
</table>
Context case study
Case study

- Jump to past (Q4 2021)
- EXA1 (new supercomputers) Lustre mount request :)
- New Atos BXI internal network (specific LND)
- 60 LNET routers (no multirail)
- Add 7K clients to existing Lustre filesystems
- Lustre/HSM case study
I want to mount Lustre FS

Lustre/HSM STORE

T1K1.2 (~3000 nodes)

T1K2.2 (~9000 nodes)

~200 Lnet routers

4x Metadata servers (MDS/MGS)
5x MDTs

40x Data servers (OSS)
384x OSTs

~60 Lnet routers

EXA1 (~7000 nodes)
Case study

- ~20K Lustre clients
- 3 supercomputers mounted (many) same Lustre FS
- Heterogeneous routers configuration (BXI_V1, BXI_V2, ConnectIB, ConnectX-4, ConnectX-6)
- All Lustre servers are Virtual Machines (limited resources)
- HA configured (7 failover nids on Lustre targets)

What we know:

- Follow best practices Lustre scales well around 10K clients (thanks to Lustre community and users feedback)
- Cray had some issues with many large supercomputers
  - [https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap166.pdf](https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap166.pdf)
- BXI_V2 pretty young: issues on large scale (fixed now)
Baby storm is here

▶ Give us a try to mount FS:
  - Unable to mount all nodes (7K) at same time (+12K already mounted)
  - Targets disconnection on already mounted clients
▶ Successfully mounted (step of max 500 parallel mounts)
  - Not very hopefully
  - Not pretty robust for future...

Analysis highly “recommended”, first issues seen:
▶ Lnet credits starvation on routers/servers
▶ Memory usage increases (50% of total memory)
▶ Many “small” RPC on MDS/MGS node (~180000 RPC/s)
Lustre layers blockdiagram

Figure 4. Basic view of Lustre software stack

- Point to point communication
- End to end communication

Extracted from excellent ORNL Lustre Internals Papers
https://info.ornl.gov/sites/publications/Files/Pub166872.pdf

Extracted from LAD’15 Chris Horn talk
Credits and peer credits
Credits and peer_credits

First analysis:
- Not enough resources on server side to absorb client activity
- Seen by monitoring Inet interface credits (peer_credits starvation)

Adjust credits/peer_credits: read Chris Horn LAD’15 talk again :)
- Credits = Throttling mechanism (a “TCP congestion window” like)
- Notice: peer_credits != credits
- Credits: Amount of credits per interface (o2ib10 here) for all CPT (CPU Partition Table)
- Peer_credits: Max “inflight” Inet messages allowed for a peer
Credits and Credits computation

- **Peer Credits:**
  - Fine tuning in Lnet routing configuration (router is the closest/direct hop)
  - Max value 255 (LND/Network driver compatibility ie max_send_wr supported), default 8
  - CEA values
    - Servers: 42 (CX Firmware + MOFED 4.7.3 limitation due to ConnectIB interfaces)
    - Clients: 42 except for BXI LND (32)
    - Routers: 42 (IB LND), 32 (BXI LND)

- **Credits:**
  - Default: 256
  - **Rule of thumb (CEA):** credits=$((peer_credits*max_peer_seen))
    - CEA credits servers=$((42*260))=10920
    - Notice: value is divided per CPT (min value: 64 per CPT)
Credits computation for routers

- **No computation needed** (of course yes!)

- **Monitoring router buffer credits needed**:  
  - Relevant only on routers  
  - Give credit live consumption per CPT  
  - for each type of RPC
    - 0: ack and control packets  
    - 1: <= 4kb size RPC (ping and others)  
    - 256: >4Kb RPC (I/O)
  - count: credits left  
  - credits: total available buffer credits  
  - min column: low water mark (negative value indicates credits starvation)

```
# cat /sys/kernel/debug/lnet/buffers
pages  count  credits  min
0      1024   1024     1021
0      1024   1024     1020
1      8192   8192     8059
1      8192   8192     8036
256    512    512      507
256    512    512      507
```
Credits tuning results

- Parallel 7K clients successfully
- Seems few disconnection on some part of 7K nodes
- Few credits starvation, lower water mark < 3 digits (no queue)
- Others clusters are fine
Still work to do

- A big job which breaks 2K nodes
  - generating BXI interconnect instability
  - some clients were able to partially communicate, some others fully hang

- **Lustre level**: catastrophic effect
  - OOM on routers and servers
  - Others supercomputers are stuck

- **Lnet storm messages**: up to 7 800 000 RPC/s from routers to servers

- How can we handle this? (we hadn't the BXI fix at this time!)
### CPU Partition Table
Handle 7 800 000 RPC/s?

- **Inet_selftest** is our best friend
- Seen a limitation on IB per servers/routers to ~180 000 RPC/s
- Non optimal CPU load distribution (pool of 4 CPUs closed to 90%, some others pools to 30%, ...), Hashing function per peers ([LU-14676](#))
- Goal: add more CPU Partition Table to have a better distribution from routers (avoid NID overlap per CPU)
- CPU server configuration: 12 cores, 1 NUMA node

```
# cat /etc/modprobe.d/libcfs.conf
```
Result: big boost to 1 800 000 RPC/s per server

Still many RPC/s on MDS/MGS node during storm (Interconnect hang)
- Move MGS outside MDS on a Virtual Machine
- Useful to raw monitor MGS traffic and of course MDS traffic

CPT live configuration:

```
lnetctl net show -v
cat /sys/kernel/debug/lnet/cpu_partition_table
```

Notice: these tunings increase significantly performance, the tradeoff is the memory usage
High priority RPC nightmare
High priority RPC nightmare

- Configuration was pretty stable but:
  - Still massive small RPC (High priority) rates in case of network failure
  - No OOM except with one bad user workload

- How can we handle this?
  - Understand the storm HP RPC rates on reconnection
  - User workload analysis

- Storm HP RPC rates:
  - Only obd_ping RPCs seen
  - Obd_ping gives health status of Lustre servers and clients in both ways
  - Obd_ping are sent every “obd_timeout/4” (“keepalive” like)
  - On hp rpc timeout, clients immediately resend an obd_ping (no more credits on routers/servers for example...)
  - On large configuration (clients+lustre targets), Lnet flooding occurs
High priority RPC nightmare

▶ OOM on bad user workload
  - After profiling, workload ~= an IOR (FPP) from 1000 nodes (128000 processes) with a striping to -1 (here 384) on each file. A bit huge...
  - OOM root cause comes from ost_io services.

▶ Solution:
  - Tell user not to retry, not a persistent solution.
  - How to fix both issues (OOM + obd_ping burst)?
Solution thanks to adaptive timeout

- Obd_ping storm comes from network issues (Lnet credits starvation or lack of resources on the path)

- For RPC timeout, Lustre relies on Adaptive timeout. Best practices often talk about at_max, not often at_min:
  - From Lustre manual “The at_min parameter is the minimum processing time that a server will report”
  - “0” means pretty fast when all is fine on Lnet network and Lustre servers/routers/clients
  - In case of “target” failure, this value is used by clients to send HP RPC to the target => flooding occurs
At_min

>We can help Lustre to not retry before Lnet message timeout
>HAVE TO RELY ON YOUR LNET CONFIGURATION:

```
# lnetctl global show
global:
    numa_range: 0
    max_intf: 200
    discovery: 0
    retry_count: 0
    transaction_timeout: 50
    health_sensitivity: 0
    recovery_interval: 1
```

>Here, no retry_count, a recommendation is to have:
>• at_min=((retry_count + 1) * transaction_timeout+”piece of processing time”)
>cea value: at_min=55
Summary
Summary

- Simulating a 80K parallel mounts clients: take 30s, credits starvation occurs, clients wait Lnet transaction timeout before to retry
- On production, 20K clients mount successful
- No more OOM in case of bad user workload or network instability
- BXI_V2 fixed now (no more full internal interconnect hang)
- No call to “ptlrpc_grow_req_bufs” function
- TBF QoS in use to limit bad user workload

- Large stable Lustre filesystem “relies” on:
  - Lnet credits tuning
  - CPT configuration adapted to your hardware
  - Adaptive timeout well tuned (obd_timeout, at_min, at_max)
Questions?
QoS Deep Dive related
Need a quick reproducer of OOM (outside production servers), how to?
- mount 20 times the same test Lustre Filesystem on 1000 nodes (goal: simulating 20K exports on servers)
- Forge bad Lnet routes (to add noise on Lnet routing). Reverse Lnet network on routers.
- monitor obd_ping, “ptlrpc_grow_req_bufs” function
  - routerstat
  - lctl dk set_param=+malloc

Events seen:
- massive High Priority RPC occurs during reconnection (due to credits + cpt tunings), obd_ping > 200 000 RPC/s
- requested buffers (rqbds) are allocated and grow up to OOM
Workaround in place

- Issue: requested buffers is the PTLRPC entry point from upper Lustre services:
  - Only low level (dangerous) tuning available
  - Trial method is an acceptable way to find upper values (not an easy task to simulate the worst case scenario on your FS, take idea from users workload)

```
# lctl get_param *.*.*.req_buffers_max
ldlm.services.ldlm_cancelrd.req_buffers_max=0
ldlm.services.ldlm_cbd.req_buffers_max=0
mds.MDS.mdt.req_buffers_max=0
mds.MDS.mdt_fld.req_buffers_max=0
mds.MDS.mdt_io.req_buffers_max=0
mds.MDS.mdt_out.req_buffers_max=0
mds.MDS.mdt_readpage.req_buffers_max=0
mds.MDS.mdt_seqm.req_buffers_max=0
mds.MDS.mdt_seqs.req_buffers_max=0
mds.MDS.mdt_setattr.req_buffers_max=0
mgs.MGS.mgs.req_buffers_max=0
```

- Setup custom req_buffers_max values for mdt,oss services