LUSTRE SERVER ON KERNEL 4.12.8

LUG 2018 – Argonne, USA | Gaël DELBARY

23 AVRIL 2018
Topics

- Flash CEA project
- Hardware Storage (for this analysis)
- Performance recorded
- Analysis
- Lustre build on kernel 4.12.8
- Results with kernel 4.12.8
Flash Parallel Filesystem (Q1 2019) for the next exascale supercomputers

Target:
- First store storage level (same Lustre FS as STORE) with Lustre Pool
- 1 TB/s in write sequential (5% of times)
- Mounted by all supercomputers
- Data migration with RobinHood (via lfs migrate)
- Hide complexity to end users

Usage:
- Checkpoint restart files (mostly sequential writes)
- Final files (mostly sequential writes)
- Data post-processing (sequential writes + random read)
Flash required?

Reasons:
- Low footprint (limit to 5 racks)
- Random read
- High write throughput

External things to monitor:
- Weight (density is not light)
- Power (SSDs consume energy)

Collaboration with many vendors

Focus on write throughput

Goal: optimize Lustre write sequential throughput on a given hardware
HARDWARE STORAGE

- Use case for this topic: embedded platform
- DDN SFA14KXE with SFAOS 11.0 GA
  - Broadwell processors
  - 512 GB of RAM
  - Infiniband Mellanox EDR
  - QEMU inside for Embedded
- 1x Declustered RAID pool (20x SSD HGST SS200)
  - 2x hotspares
  - 18x usable disks (RAID5 inside)
  - Theoretical write sequential throughput (~16GB/s)
- Tested configuration: 1 VM (OSS inside)
  - 90 GB of RAM
  - 8x CPU cores (16 virtual with hyperthreading)
  - 2x EDR Infiniband card (Full Bandwidth: 24 GB/s)
Commons parameters:

- Device formated with no lazy initialization
- CentOS 7.3 (ioscheduler: deadline, tuned profile: network-latency)
- MOFED 4.2.1
- Lustre 2.10.2 (ldiskfs backend)
- Lustre clients tuning (no checksums)
- FIO (direct=1, ioengine=libaio, iodepth=1)
- IOR 3.0.1 (8 EDR clients)
Goal remember: close to 12 GB/s (full fill one EDR interface) with IOR

Bandwidth lost:
- 8% Theoretical vs Ext4
- 11% Ext4 vs Ldiskfs
- 19% Ldiskfs vs IOR

IOR bottleneck?
- EDR bottleneck: 12 GB/s
- Still 12,5% IOR vs EDR

Write sequential throughput

Centos7 (Lustre 2.10.2)
Not EDR bottleneck, Inet_selftest (write) = 11.7 GB/s

OSS Load is pretty low

Kswapd is 100% CPU => memory reclaim?

Behavior: full throughput until kswapd starts its job

Need a quick profiling
Not enough free pages available => switch to direct reclaim mode

Direct reclaim mode is “time” expensive (due to spin_lock mechanism)

1/3 times in memory allocation through page cache

This time could be use to run I/O

What we have benchmarked in reality?

- The speed of writing data in page_cache and flushing in the backend disks
- Means page cache throughput limits to 10,5 GB/s (on this VM and on this hardware)?
IMPROVEMENTS TO STUDY

- Bypass page_cache for Lustre OSD-layer: need big stuff in Lustre (discussion with Intel)
- Improve page_cache mechanism in Kernel: a delicate topic (seems nobody wants to do some modifications)
- What we try to do: decrease the latency to lower layer to expect benefits in throughput
  - Layer retained: io scheduler
  - From kernel 3.13, full io bypass scheduler exists
  - To get latest modifications to ext4, try 4.12.8 kernel (current version when we did tests)
io scheduler=none

Performance is closed to the disks backend

Improvements are amazing

Jump to port Lustre on this kernel (4.12.8) (ldiskfs backend)
2 main jobs (ldiskfs backend):

- Ldiskfs (34 patches, ~ 8655 lines)
- Osd-ldiskfs

LU-9558 to support Lustre “Client” Layers on kernel up to 4.13 (nice job)

Process: build and fix errors until success (slow but give results…)

Simplification: disable features that not build and not critical (gss, project quota, ext4 encryption)

Ldiskfs current build process (main steps):

- Copy of ext4 sources from current kernel to a temporary folder
- Apply some patches for the current kernel (through series folder)
- Sed “ext4” by “ldiskfs”
Exists for Redhat/CentOS kernel (3.10.x), a bit old for 4.12.8!

Exists for Suse kernel (4.4.x), interesting because difference is less than previous

Create a custom series based on Suse ldiskfs patches

First pass was done with patches for Suse kernel on kernel 4.12.8

- 10/34 patches have success with no modification
- 13 patches to modify

Good surprise: some ldiskfs features has been landed in ext4 vanilla kernel (thanks to people that does this work): some patches are useless (11, that’s huge)

Final verification and round-trip with Redhat kernel series (sometimes 4.12.8 is closed to 3.10 for some ldiskfs features)
Ldiskfs build pass for 4.12.8 kernel, osd-Ldiskfs part?

No surprise, failed on build, minor modifications needed:

- “PAGE_CACHE_SHIFT” doesn’t exist anymore, just renamed in “PAGE_SHIFT”
- Same way for “mutex_lock” (mechanism still exists, another way to do the job with “inode_unlock”)
- Inode struct has changed, new way to do xattr operations. “i_op->*xattr” is gone. “__vfs_*xattr” is an alternative

Not too much hard job, kernel commits help to understand the new way

3 new patches for osd-Ldiskfs (222 lines)
LUSTRE ON KERNEL 4.12.8: MOMENT OF TRUTH

- Full Lustre rebuild (--disable-quota --disable-gss --with-o2ib=yes) with MOFED 4.2.1
  - OST format step: Pass
  - Ldiskfs mount: Pass
  - Lustre mount: Crash (Null pointer)

- Crash analysis:
  - “ext4_dir_operations” struct changes in kernel 4.12.8
  - Operation vector “.iterate” moved to “.iterate_shared”
  - “osd_scrub.c” and “osd_handler.c” want “.iterate”

- Fix was done through a patch in ldiskfs tree (no modification in osd part)
- This lustre version is used on a flash filesystem mounted on a lab supercomputer from 4 months with no crash. (that’s not sufficient to proove its stability…)
Ldiskfs throughput is very close to ext4.
IOR is around 11.7 GB/s, the EDR interface is full filled.
Page_cache behavior has disappeared or offset?
Multirail bench to remove limit to 12 GB/s?
2x EDR interfaces on the OSS

8x clients

Result: 12.7 GB/s

Where is the bottleneck?

Page_cache again?
Page_cache behavior has just shifted to upper value

Peek performance (15.7 GB/s) close to backend until memory reclaim occurs

Quick explanation (to be confirmed): server memory bandwidth is reached during memory reclaims
Page_cache usage has impact on Lustre performance

A workaround is to reduce latency in some layer:
- to flush quicker data on disks
- Effect will be to liberate memory pages soon
- Memory allocation will be faster
- I/O throughput will increase

For example: NVMe disks (latency gain) but cost is high (now)

Best performance (with low latency disks) will need a full page_cache bypass

TODO:
- Add patches to LU-10942 (cleaning in autoconf, add “define” in lustre source for the kernel 4.12.8)
- Add support for features (GSS, Project Quota) not working yet
- Add support for next Redhat 8 kernel
Questions?