FSx for Lustre

FSx for Lustre and Open Source

Timothy Day

(He/Him)

Topics

01 Overview

02 Elastic Fabric Adapter (EFA) for Lustre

03 LLVM and Static Analysis

04 In-memory OSD

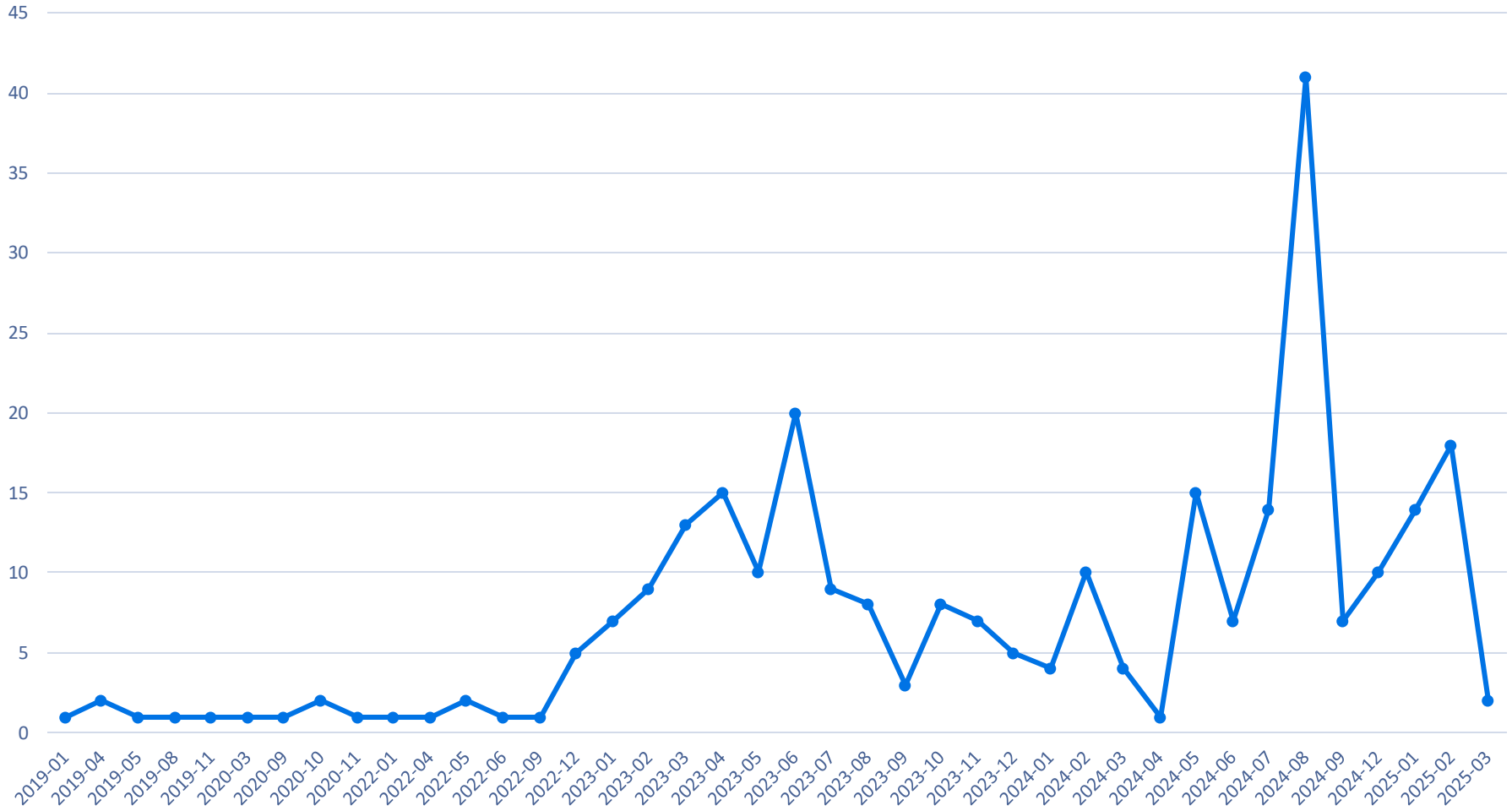
05 Upstreaming to Linux and LSF

06 Future Work

Overview

Contributions from AWS

NUMBER OF COMMITS CONTRIBUTED IN EACH MONTH



High Activity

AWS is increasingly active in Lustre.

Generated with:

```
git log --pretty=format:"%ae %ad" --
date=format:"%Y-%m" | grep "@amazon.com" |
awk '{print $NF}' | sort | uniq -c
```

Elastic Fabric Adapter (EFA) for Lustre

Why EFA?

- Unlock higher network throughput for customers
  - 8x – 12x throughput improvement on p5 H100 GPU instances
  - GPUDirect Storage (GDS)
  - RDMA
- Lustre is designed around RDMA

Protocol and LNET Changes

- LU-18808 - Unique addressing format for 2.15+ (to avoid IPoIB)
  - 171.12.97.0@efa = 10.200.171.12@tcp + 61:00.0
- Heterogeneous network
  - Protocol negotiation
  - MDS on ksocklnd and OSS on kefalnd
- Leverage UDSP and LNET Multirail

Future Work (EFA)

- Upstreaming!
- Investigate new p2pdma APIs (565.57.01) for GDS on for upstream Linux client

LLVM and Static Analysis

The Strengths of LLVM

- LU-16518 - Implement LLVM/Clang build support
- Different set of default warnings
- Powerful plugin ecosystem
  - LU-8191 – Marking static functions with FindStatic
  - LU-18753 - Purging orphaned code with xunused
  - Enforcing code organization
- Many opportunities for introspection

How to Use It

- Build the kernel using LLVM: https://docs.kernel.org/kbuild/llvm.html
- Build Lustre using LLVM:

./autogen.sh

./configure LLVM=1 – OR –

./configure LLVM=1 --disable-strict-errors

make

Everything should just work!

LLVM vs. Alternative Static Analysis

- LU-17000 – Fix Coverity Scan issues
- Coverity captures many issues
- Coverity has a higher false positive rate
- LLVM and plugin ecosystem is FOSS and easier to run locally
- LLVM is extensible

Future Work (LLVM)

- LLVM19+
- Leverage plugins to enforce code organization efforts required by upstreaming
- Improve native plugin integration

In-memory OSD and more!

In-memory OSD

- LU-17995 – Implemented a purely in-memory OSD (Object Storage Device). This enables Lustre servers to be run solely from memory (without ZFS or ldiskfs)
  - Prototype (3-5 kLoC)
  - Primarily targeting testing use-cases (at first)
- LU-17848 – Share more code between existing OSDs
- LU-17079 – Improve userspace handling of new OSDs
- LU-18813 (DDN) – OSD wbcfs derived (partially) from the in-memory OSD prototype and the memfs from MetaWBC

Future Work (OSD)

- Land the initial implementation of the OSD
- Enhance debugging/monitoring infrastructure
- De-duplicate more code!
- Update the documentation in “Documentation/osd-api.txt”
- Investigate mirroring or writeback

General Improvements

- LU-8802 – Remove MAX_OBD_DEVICES to allow many mounts
- LU-18162 – Improve lu/obd device handling
- LU-17216 - Tunable enable_health_write disk health checker
- LU-17242 - General debugging improvements
- LU-17862 - Unify more kernel modules

Upstreaming to Linux and LSF/MM/BPF

Upstreaming Lustre into the Linux kernel

- Good for Lustre
- Benefits AWS customers
- Lustre and Linux are stronger together!
- Much of our work is aimed towards upstreaming

LSF/MM/BPF

- Attended LSF/MM/BPF to advocate for upstreaming Lustre into the Linux kernel
- Warm reception!
- Plenty of work remaining:
  - Focusing primarily on the client
  - We must convert Lustre to use folios!
  - Client/Server code split
- Follow the project on the newly created wiki page: https://wiki.lustre.org/Lustre_Upstreaming_to_Linux_Kernel

Thank you!

Timothy Day

timday@amazon.com