BGI Research 华大生命科学研究院

兜兜转转, 回到原点 After Exploring Many Options

Lustre为何是最终选择
The Answer Remains Lustre

李焱 Homer Li 2025-10

History

200X

2003 Lustre 1.0

2005 FUSE, glusterfs

2007 NFSv3 in Linux, HDFS

2008 Beegfs,ceph

201X

2012 S3FS,Alluxio,glustrefs

2015 SeaweedFS

2016 Minio

2017 wekaIO

2019 Cubefs

202X

2021 GooseFS, Juicefs

Lustre, GPFS, BeeGFS, and WekaIO. These are the powerful, in-kernel parallel file systems, designed for one thing: maximum speed in traditional supercomputing

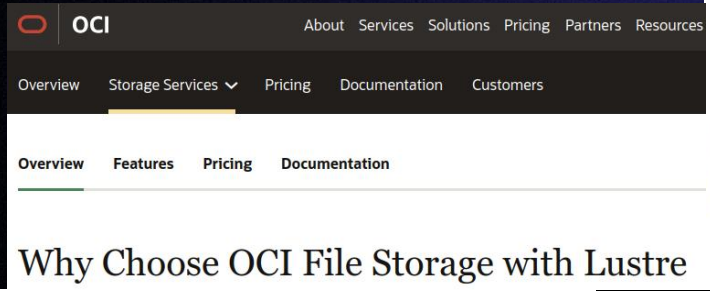
But then, as the demand for data from AI and ML grew stronger, the world changed. A new problem emerged: how to connect our traditional applications to the massive, scalable world of object storage

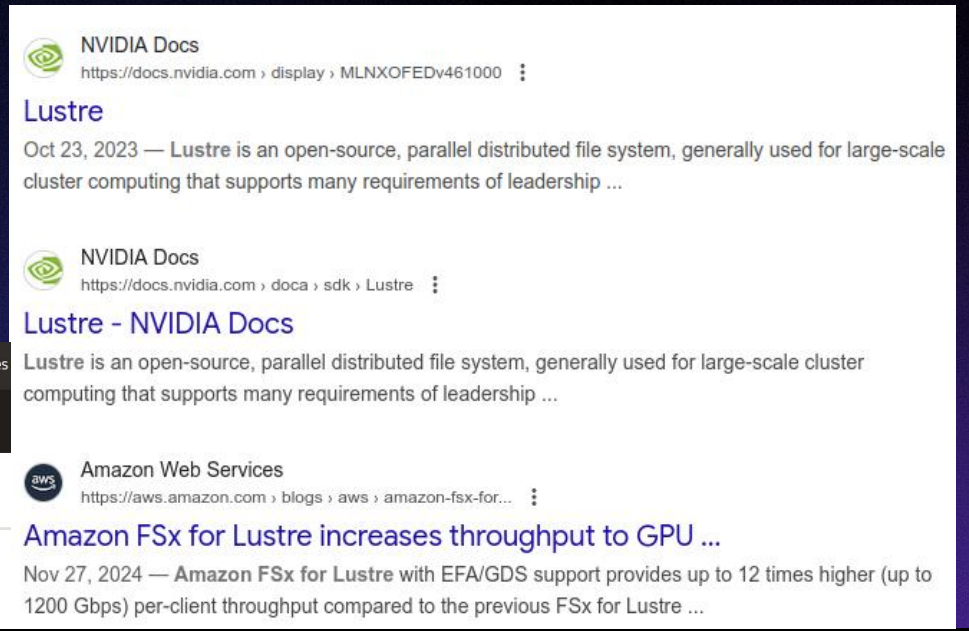
S3FS is first bridge

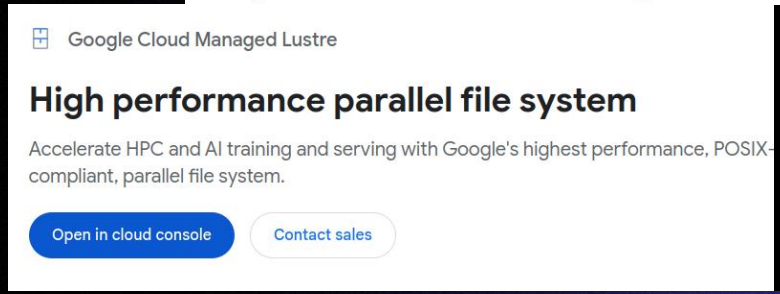
Compare the old giants with these new, flexible systems

廉颇老矣

Some friends asked me: Is anyone still using Lustre? Is this filesystem obsolete?



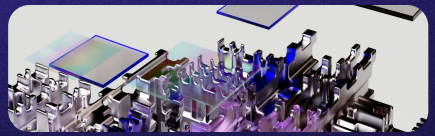






Compare with these FS based FUSE

Every gift bestowed by fate has a price secretly attached



Semantics

- Ctime/mtime/atime
- Hard link
- no atomic renames
- Append failed
- Stat/statfs
- Fsync != sync to object
- ACL, attr, xattr



Consistent

- Strong consistent
- Close to open consistent



IO

- Object storage use of large block file for small read/write
- RMW in partial changes
- In dataloader step, import different small files
- prefetching fail within large file
- Deeper call stacks, higher latency



MDS

- NoSQL default setting will cause the metadata lost
- Metadata lost cause object leak

Why test ?

We have evaluated numerous offerings from various cloud providers with the objective of identifying one or more viable, cost-effective cloud object storage solutions

I'm not saying FUSE-based FS can't do these things. It depends on the product's design goals, whether it needs to handle these scenarios, and it's a matter of maturity.

And I'm not saying FUSE-based FS is unusable. It's just that I've seen too many 'power fantasy' stories, and a little balance from the other side is needed.

FUSE-based FS can effectively simulate filesystem on top of object storage, and it provides throughput and some of file system features

FUSE interfaces are defined by their flexibility, which unlocks a world of creative possibilities.



A mechanical door handle can be a lifesaver sometimes

e.g. why is Erasure Coding not a critical feature for Lustre?

Some FUSE-based FS have incomplete basic functionality

```
root@iv-ye4we3bfuowh2yplla5d:/mnt# date +%s; for i in {0..99}
> do
> cat 1G.file >> 100G.file
> done ; date +%s
1759127236
cat: write error: Input/output error
cat: write error: Input/output error
cat: write error: Input/output error
cat: write error: Input/output error
cat: write error: Input/output error
```




```
fio: io_u error on file /mnt/rw.0.0: Permission denied: write offset=93323264, buflen=1048576
fio: io_u error on file /mnt/rw.0.0: Permission denied: write offset=94371840, buflen=1048576
fio: io_u error on file /mnt/rw.0.0: Permission denied: write offset=95420416, buflen=1048576
fio: io_u error on file /mnt/rw.0.0: Permission denied: write offset=96468992, buflen=1048576
fio: io_u error on file /mnt/rw.0.0: Permission denied: write offset=97517568, buflen=1048576
```

```
mkdir: cannot create directory ‘404ffdbd523c7ac6ebfba6deb8ab7c41/0/testdir_9/00d45fe6_0/c2774a97_1/ddb8a6
88_2/67095490_3/6be10920_4/eb458d22_5/6a7d6b0e_6/31fc8d5a_7/d8c33081_8/57421ffb_9/ec87c648_10/1f41d647_11
/5acb723d_12/9f04a0c2_13/c50aa900_14/18c14b13_15/b851eca5_16/f1ccebb7_17/8872e035_18/6c0a1c46_19/a4d55d71
_20/e2837fc1_21/722ebd86_22/35e5611b_23/8eb28333_24/29d712cc_25/4b191bc8_26/6e7072f5_27/0ab6df44_28/3135c
86d_29/f2baad4b_30/29bbab2c_31/e106bc2e_32/0e4fd6c7_33/bdddb4aa_34/56c84624_35/ccf370cf_36/925564ec_37/f1
6332de_38/59191a6b_39/a6145ac9_40/101634e2_41/a7557208_42/d4e2d445_43/bc8da6e3_44/7f58b39b_45/531116d2_46
/e8ce8324_47/b6768767_48/ae12595e_49/066c8877_50/01edf5a5_51/ec124fe4_52/5e2eee24_53/daa4c652_54/fa38757e
_55/08610271_56/f8b944be_57/97295328_58/20f1891e_59/7583e2b5_60/085a5734_61/8d0ff22e_62/3b3e1309_63/db95c
506_64/2e3773fa_65/b5579d79_66/4fdb7e08_67/03ac5c44_68/c4f1fc24_69/7db81c71_70/aefda8ba_71/e4c8e9a0_72/1c
c9aa95_73/3e5c92a8_74/59e467c0_75/cb7fe9db_76/beea7d66_77/97a68855_78/4db5a3ff_79/32c12fd8_80/2c7f4b0d_81
/9a2343fa_82’: Operation not supported
```



Unfair benchmark for Lustre

Cloud or Some FUSE-based FS vendors are waving around their benchmark results, bragging about the high scores they got.

Eventual consistency VS Strict consistency

Local SSD R/W cache VS Lustre not add local cache

Custom semantics VS Traditional file system semantics

Test info

1x Xeon 4314
8x 32G DDR4 2666MHz
2x 25GbE ConnectX-4 Lx
8x KIOXIA KPM61VUG1T60; 4x Lustre OST(no raid), 4xMinio (EC:0)
OSS Lustre 2.15.7, AlmaLinux 8.10, minio RELEASE.2025-04-22T22-12-26Z, ext4

1x Xeon 4314
8x 32G DDR4 2666MHz
2x 25GbE ConnectX-4 Lx
2x KIOXIA KPM61VUG1T60; 1x Lustre MDT(no raid, ldiskfs), 1xReids
MDS Lustre 2.15.7, AlmaLinux 8.10, redis 7.0.8, ext4

1x Xeon 4314
8x 32G DDR4 2666MHz
2x 25GbE ConnectX-4 Lx
Client, Lustre 2.15.7 AlmaLinux 9.5, latest FUSE-based FS, default parameters, enable full cache for best performance

FUSE-based FS test issues

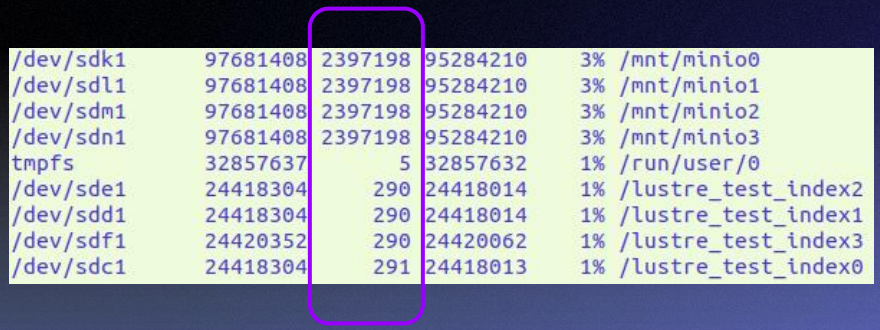
- Big block(1MiB or 4MiB) in object storage
  - RMW, COW no helps
    - Object storage Merge
    - FUSE-based FS client read write SSD cache to upload
    - FIO random write test

FIO randwrite 128KiB test, trace the read

OSS read in random write
Process minio (665007) is reading file part.1 reading size: 32, fsize: 1048704
Process minio (665007) is reading file part.1 reading size: 262144, fsize: 1048704
Process minio (665007) is reading file part.1 reading size: 32, fsize: 1048704
Process minio (665007) is reading file part.1 reading size: 262144, fsize: 1048704

Client read in random write
Process xxxfs (4372) is reading file 57457953_15_4194304 read size: 360448,fsize: 4194816
Process xxxfs (4372) is reading file 57457953_15_4194304 read size: 44,fsize: 4194816
Process xxxfs (4372) is reading file 65894459_0_4096 read size: 4096,fsize: 4100
Process xxxfs (4372) is reading file 65894459_0_4096 read size: 4,fsize: 4100

FUSE-based FS test issues



/dev/sdk1	97681408	2397198	95284210	3%	/mnt/minio0
/dev/sdl1	97681408	2397198	95284210	3%	/mnt/minio1
/dev/sdm1	97681408	2397198	95284210	3%	/mnt/minio2
/dev/sdn1	97681408	2397198	95284210	3%	/mnt/minio3
tmpfs	32857637	5	32857632	1%	/run/user/0
/dev/sde1	24418304	290	24418014	1%	/lustre_test_index2
/dev/sdd1	24418304	290	24418014	1%	/lustre_test_index1
/dev/sdf1	24420352	290	24420062	1%	/lustre_test_index3
/dev/sdc1	24418304	291	24418013	1%	/lustre_test_index0

Inode used number

- Merge or Split ?

After system initialization, writing over a dozen 100 GiB files caused them to be segmented into numerous small objects on the storage backend.

This process generated a lot of directories and sharding files, leading to significant metadata pressure when storing large files at scale.

Additionally, the Lustre file system has the same dir/file structure

Close to open consistency I

Simulate two different nodes for both FUSE-based FS and Lustre clients by mounting the same remote filesystem to different local directories on the same client machine

Eg:
mount.lustre xx.xx.xx.xx@tcp:/lustre /mnt/lustre-1 -o localflock,user_xattr
mount.lustre xx.xx.xx.xx@tcp:/lustre /mnt/lustre-2/subdir -o localflock,user_xattr

lustre-1/subdir/test-file is lustre-2/test-file
fusefs-1/test-file is fusefs-2/test-file

Test workflow:

1. Write/append data from /mnt/lustre-1/test-file
2. Close /mnt/lustre-1/subdir/test-file
3. After Close return OK
4. To Read file tail (tail -c xx bytes) from /mnt/lustre-1/test-file and /mnt/lustre-2/test-file
5. Compare the cksum, if cksum != , return failed

Lustre and nfs(async) always returns a matching checksum
FUSE-based FS failed with cache

```
3
9+0 records in
9+0 records out
20322 bytes (20 kB, 20 KiB) copied, 0.0152072 s, 1.3 MB/s
630 -rw-r--r-- 1 root root 645051 Oct 11 15:34 /mnt/fusefs-1/test-file-1234
630 -rw-r--r-- 1 root root 645051 Oct 11 15:34 /mnt/fusefs-2/test-file-1234
hash1:03aa01e41656272d32a6a46a591211262a95e8949d7fac019ba36b831b362f60  -, 645051 bytes
hash2:03aa01e41656272d32a6a46a591211262a95e8949d7fac019ba36b831b362f60  -, 18541268 bytes
The content is consistent
4
11+0 records in
11+0 records out
38368 bytes (38 kB, 37 KiB) copied, 0.0145277 s, 2.6 MB/s
28968 -rw-r--r-- 1 root root 29662385 Oct 11 15:34 /mnt/fusefs-1/test-file-1234
28968 -rw-r--r-- 1 root root 29662385 Oct 11 15:34 /mnt/fusefs-2/test-file-1234
hash1:4261a7f63fb96392da6a80b4b9948896f2c5dfcf9e21a8d03fd442faefa19928  -, 29662385 bytes
hash2:4261a7f63fb96392da6a80b4b9948896f2c5dfcf9e21a8d03fd442faefa19928  -, 645051 bytes
The content is consistent
5
14+0 records in
14+0 records out
22974 bytes (23 kB, 22 KiB) copied, 0.018678 s, 1.2 MB/s
26357 -rw-r--r-- 1 root root 26988856 Oct 11 15:34 /mnt/fusefs-1/test-file-1234
26357 -rw-r--r-- 1 root root 26988856 Oct 11 15:34 /mnt/fusefs-2/test-file-1234
hash1:633aaecf9969ad935585dd6179146d26068b81f3ee6b01873fafa2979e65d4a7  -, 26988856 bytes
hash2:633aaecf9969ad935585dd6179146d26068b81f3ee6b01873fafa2979e65d4a7  -, 29662385 bytes
The content is consistent
6
12+0 records in
12+0 records out
19272 bytes (19 kB, 19 KiB) copied, 0.0244261 s, 789 kB/s
 7922 -rw-r--r-- 1 root root  8111767 Oct 11 15:34 /mnt/fusefs-1/test-file-1234
26357 -rw-r--r-- 1 root root 26984448 Oct 11 15:34 /mnt/fusefs-2/test-file-1234
hash1:1fdf1224c5cd233eaeefdbf3ae6f76200577f8f22c7ecddbd4cb5ff569971799  -, 8111767 bytes
hash2:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  -, 26988856 bytes
Read BAD content in 2 files
```

Close to open consistency II

Lustre and nfs(async) always returns a matching checksum

Test workflow:

1. Remove /mnt/lustre-1/subdir/test-file (destroy cache)
2. Write/append data from /mnt/lustre-1/test-file
3. Close /mnt/lustre-1/subdir/test-file
4. After Close return OK
5. To Read file tail (tail -c xx bytes) from lustre-1/subdir/test-file and lustre-2/test-file
6. Compare the cksum, if cksum != , return failed

```
0
remove /mnt/fusefs-1/test-file sccuessful
18+0 records in
18+0 records out
6066 bytes (6.1 kB, 5.9 KiB) copied, 0.0172748 s, 351 kB/s
hash1:d360d4616ccf4b0e7091f6517550d9e1062f60c767d20b2c28e7f9993d0d38af  -, 20691360 bytes
hash2:d360d4616ccf4b0e7091f6517550d9e1062f60c767d20b2c28e7f9993d0d38af  -, 20691360 bytes
The content is consistent
1
remove /mnt/fusefs-1/test-file sccuessful
13+0 records in
13+0 records out
39351 bytes (39 kB, 38 KiB) copied, 0.0278839 s, 1.4 MB/s
 5842 -rw-r--r-- 1 root root  5981958 Oct 11 15:56 /mnt/fusefs-1/test-file
20207 -rw-r--r-- 1 root root 20691360 Oct 11 15:56 /mnt/fusefs-2/test-file
hash1:f365b8a52fa1e2dde3bbc86615b420fcad6a251d89b1d2fcd9117ccf9262bf80  -, 5981958 bytes
hash2:d360d4616ccf4b0e7091f6517550d9e1062f60c767d20b2c28e7f9993d0d38af  -, 20691360 bytes
Read BAD content in 2 files
```

Some of FUSE-based FS failed with cache

FUSE based FS disable cache is OK, And the performance degrade many times

For all performance tests, the FUSE cache was enabled

Strict consistency

The same mount type, Simulate writing to and reading from some transaction database

Testflow:

1. Launch 2 threads in parallel, writer(T1) and reader(T2)
2. T1 Generate random data
3. T1 append data to /mnt/lustre-1/subdir/test-file, close file, not read any data
4. T2 open lustre-1/subdir/test-file and lustre-2/test-file
5. T2 seek the same location, and to read append data by T1 from lustre-1 and lustre-2 and sometimes T2 reading from the file before the T1 close
1. Cksum the append data from the 2 mount point

Only lustre passed
All(I tested) FS baes Fuse enable cache, failed
All(I tested) FUSE-based FS disable all cache, failed
NFS async failed

Fsync worked ?

Only Lustre and passed in default setting
The others failed

dd if=/dev/urandom of=file oflag=sync/conv=sync

The fsync operation on FUSE-based FS I've encountered is a lie. Some only write to the local cache, some still lose data, and none of them can truly fsync the data all the way to object storage.

Fsync requires guarantees and confirmations from every layer of the software and hardware stack. For such a complex call

What fairness is there in just running a benchmark to get a high score?
Here is single feature, not all !

Before power reset , list all files
After fsync and list, power reset MDS/T and OSS/T right now

```
[root@localhost-client 22:25 /mnt/lustre/fsync]# sh fsync.sh
0+1 records in
0+1 records out
13 bytes copied, 0.00302508 s, 4.3 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00171421 s, 7.6 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00181039 s, 7.2 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.0017994 s, 7.2 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00202227 s, 6.4 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00148666 s, 8.7 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00155507 s, 8.4 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.0013763 s, 9.4 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00139155 s, 9.3 kB/s
0+1 records in
0+1 records out
13 bytes copied, 0.00163048 s, 8.0 kB/s
total 48K
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:23 file.0
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.0
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.1
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.2
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.3
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.4
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.5
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.6
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.7
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.8
4.0K -rw-r--r-- 1 root root  13 Oct 12 22:25 fsync.9
4.0K -rw-r--r-- 1 root root 151 Oct 12 22:25 fsync.sh
[root@localhost-client 22:25 /mnt/lustre/fsync]#
```

Fsync worked ?

```
total 12K
4.0K -rw-r--r-- 1 root root   7 Sep 23 22:17 file.0
4.0K -rw-r--r-- 1 root root   7 Sep 23 22:18 fsync.sh
4.0K -rw-r--r-- 1 root root   7 Sep 23 23:21 fsync.0
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.1
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.2
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.3
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.4
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.5
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.6
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.7
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.8
   0 -rw-r--r-- 1 root root   0 Sep 23 23:21 fsync.9
```

```
total 8.0K
4.0K -rw-r--r-- 1 root root  13 Oct 12 23:21 file.0
4.0K -rw-r--r-- 1 root root 108 Oct 12 23:39 fsync.sh
```

The metadata was preserved, the first file synced, the other were empty

All files written with fsync were lost

A few other things

- MDS database lost data cause object leak
  - And object leaks appear in some complex case
  - MDS config a fully consistent write mode leads to performance degraded
  - Read/Write error
- Lustre uses direct/fadvise with strict semantics, but some of FUSE-based FS does not. The system calls for FUSE-based FS still involve a lot of caching between the client and the object storage.
  - *Is this kind of test fair?*
  - *Even without cheating, Lustre's performance remains superior*

```
root@testuser-KVM:/mnt/fuse-testfs-1# date +%s; dd if=/dev/urandom of=/mnt/fuse-testfs-1/test-123 bs=128K count=40000; date +%s
1756191948
dd: error writing '/mnt/fuse-testfs-1/test-123': Bad file descriptor
dd: closing output file '/mnt/fuse-testfs-1/test-123': Bad file descriptor
1756191994
```

```
root@testuser-KVM:~# date +%s; dd if=/mnt/fuse-testfs-1/test-123 of=/dev/null bs=1M; date +%s
1756191959

dd: error reading '/mnt/fuse-testfs-1/test-123': No such file or directory
124+0 records in
124+0 records out
130023424 bytes (130 MB, 124 MiB) copied, 34.9487 s, 3.7 MB/s
1756191994
```

MDS crash
Read a file not exist

How do we handle these orphan objects that are unknown to the MDS?
Object Storage has already allocated the space

Random IO

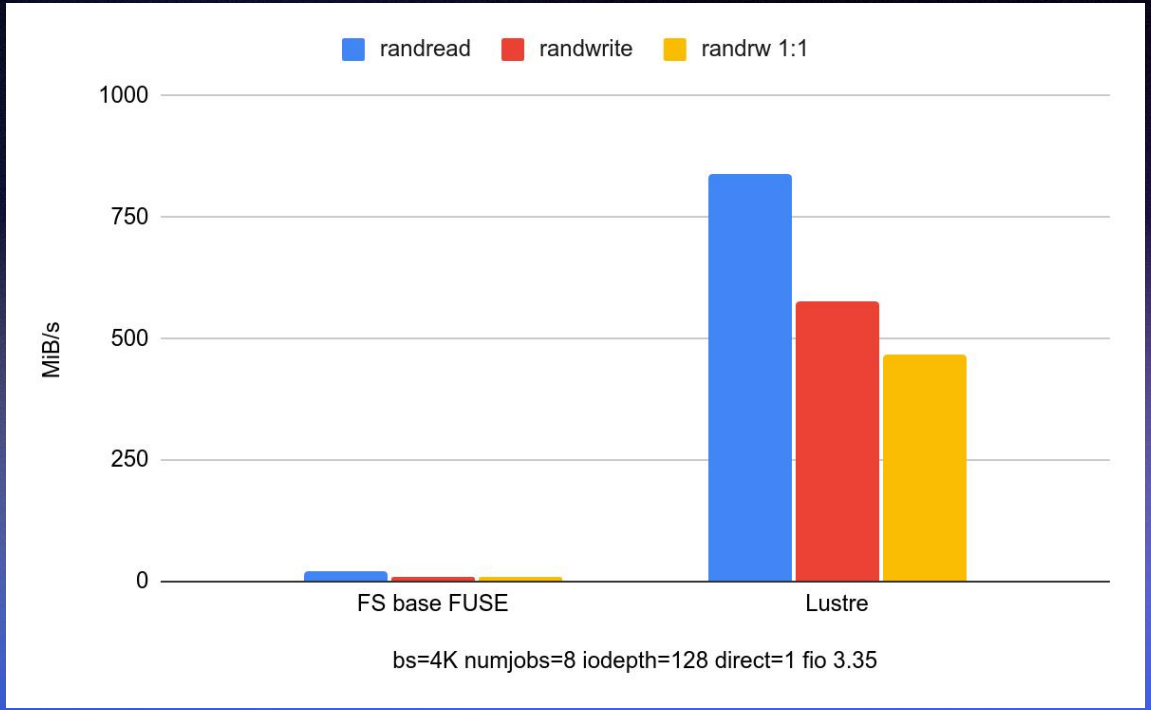
FIO test
[global]
rw=randrw
bs=4K
direct=1
numjobs=8
iodepth=128
ioengine=libaio
group_reporting
filesize=99G

[rw]
directory=/mnt

Default mount, with full cache

Lustre is 40 to 60 times more powerful !

Key Performance of a Dataloader Workflow



Metadata

Single client 8 x process test metadata
7.2M files, each dir contains 20K files

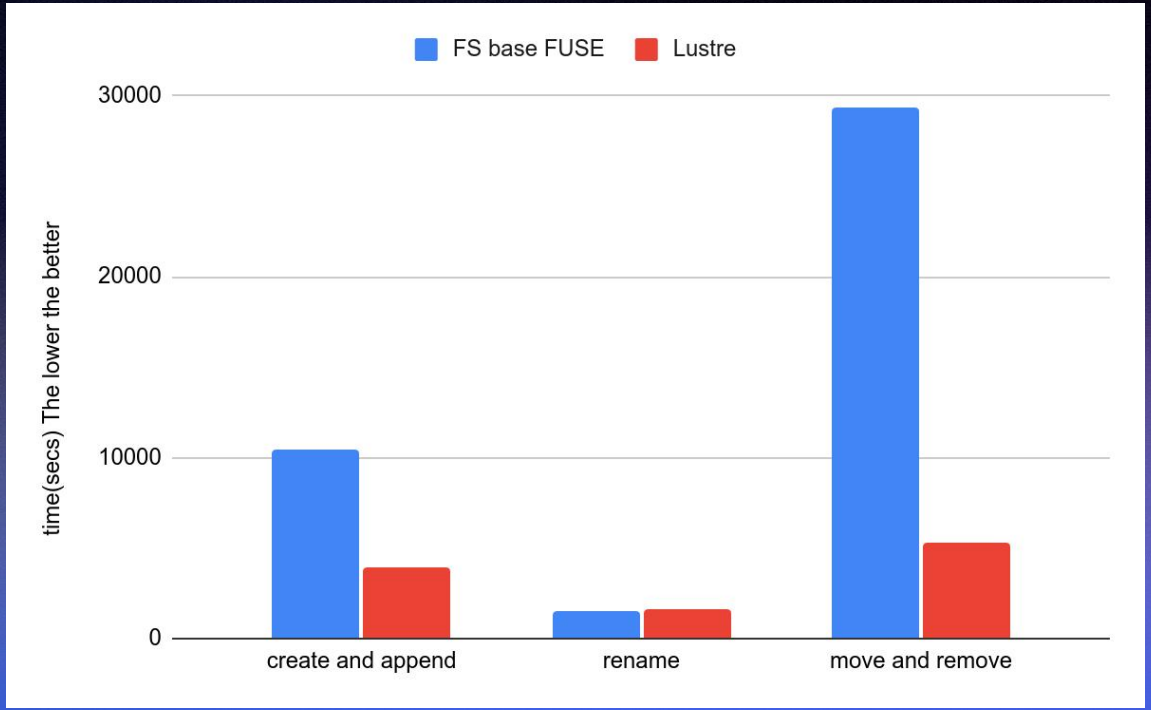
- Test workflow
  - Create file and write data
  - Append data
  - Rename
  - Move
  - Remove

Lustre 2.65x lead in create and append

Lustre outperforms the FUSE-based FS by 5.55 times in this most complex part(move and remove)

FUSE-based FS is 1.06x lead in rename test case

Default mount, with full fuse cache



Latency compare(mv in the same metadata test)

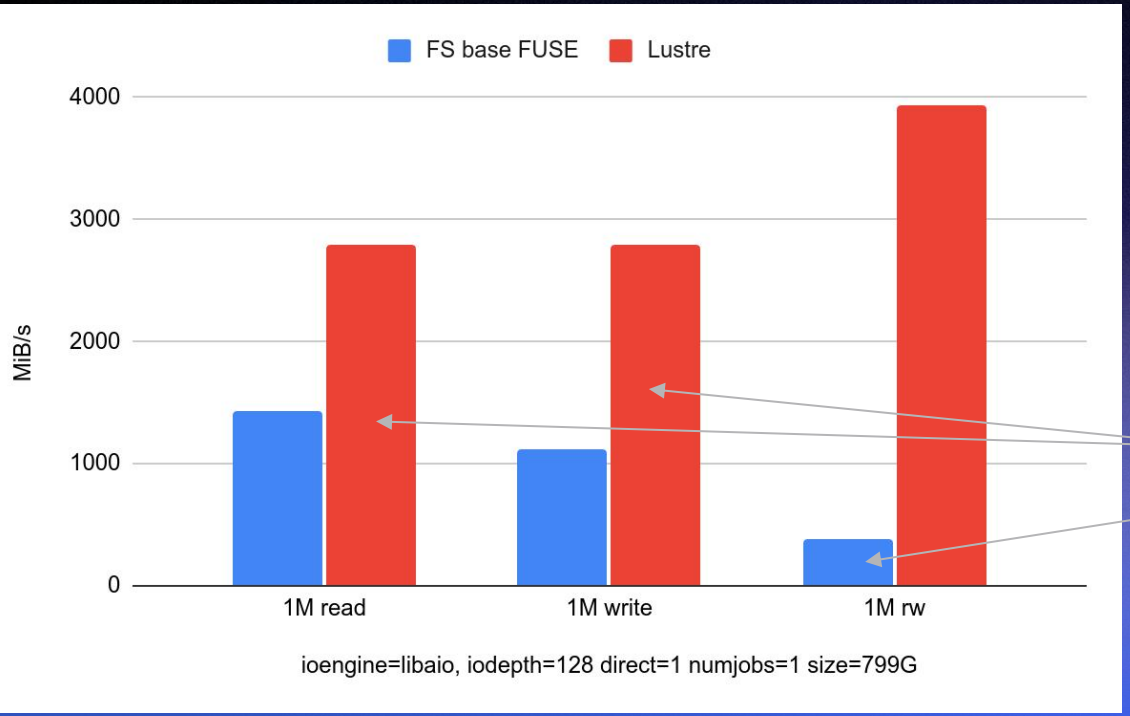
```
465.040 ( 3.052 ms): renameat2(olddfd: CWD, oldname: 0xa2939cc4, new
468.095 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
468.102 ( 2.645 ms): renameat(olddfd: CWD, oldname: 0xa2939cc4, newd
470.750 ( 0.225 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
470.982 ( 3.330 ms): renameat2(olddfd: CWD, oldname: 0xa2939d4b, new
474.315 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
474.321 ( 2.567 ms): renameat(olddfd: CWD, oldname: 0xa2939d4b, newd
476.892 ( 0.206 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
477.104 ( 3.978 ms): renameat2(olddfd: CWD, oldname: 0xa2939dd2, new
481.086 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
481.092 ( 2.809 ms): renameat(olddfd: CWD, oldname: 0xa2939dd2, newd
483.904 ( 0.212 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
484.122 ( 2.599 ms): renameat2(olddfd: CWD, oldname: 0xa2939e59, new
486.723 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
486.730 ( 2.509 ms): renameat(olddfd: CWD, oldname: 0xa2939e59, newd
489.243 ( 0.209 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
489.457 ( 2.961 ms): renameat2(olddfd: CWD, oldname: 0xa2939ee0, new
492.422 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
492.428 ( 2.728 ms): renameat(olddfd: CWD, oldname: 0xa2939ee0, newd
495.159 ( 0.263 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
495.428 ( 3.528 ms): renameat2(olddfd: CWD, oldname: 0xa2939f67, new
498.959 ( 0.005 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
498.966 ( 2.649 ms): renameat(olddfd: CWD, oldname: 0xa2939f67, newd
501.618 ( 0.209 ms): newfstatat(dfd: CWD, filename: 0x16aebea0, stat
501.832 ( 4.158 ms): renameat2(olddfd: CWD, oldname: 0xa2939fee, new
```

Lustre

```
 412.303 (102.469 ms): renameat2(olddfd: CWD, oldname: 0xd779aeab, newd
 514.776 ( 0.150 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
 514.931 (103.192 ms): renameat2(olddfd: CWD, oldname: 0xd779af33, newd
 618.126 ( 0.189 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
 618.319 (105.950 ms): renameat2(olddfd: CWD, oldname: 0xd779afbb, newd
 724.273 ( 0.313 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
 724.590 (88.464 ms): renameat2(olddfd: CWD, oldname: 0xd779b043, newdf
 813.058 ( 0.157 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
 813.219 (117.972 ms): renameat2(olddfd: CWD, oldname: 0xd779b0cb, newd
 931.195 ( 0.333 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
 931.535 (88.924 ms): renameat2(olddfd: CWD, oldname: 0xd779b153, newdf
1020.463 ( 0.328 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1020.795 (117.510 ms): renameat2(olddfd: CWD, oldname: 0xd779b1db, newd
1138.309 ( 0.228 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1138.541 (104.561 ms): renameat2(olddfd: CWD, oldname: 0xd779b263, newd
1243.105 ( 0.264 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1243.378 (119.373 ms): renameat2(olddfd: CWD, oldname: 0xd779b2eb, newd
1362.754 ( 0.237 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1362.995 (119.756 ms): renameat2(olddfd: CWD, oldname: 0xd779b373, newd
1482.754 ( 0.283 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1483.042 (117.106 ms): renameat2(olddfd: CWD, oldname: 0xd779b3fb, newd
1600.151 ( 0.432 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1600.591 (119.872 ms): renameat2(olddfd: CWD, oldname: 0xd779b483, newd
1720.467 ( 0.192 ms): newfstatat(dfd: CWD, filename: 0x7b57aea0, statbu
1720.663 (118.575 ms): renameat2(olddfd: CWD, oldname: 0xd779b50b, newd
```

FUSE-based FS

Throughput



Because Lustre is limited by the performance of a single 25GbE network port, Lustre can only reach this value.

Here is a simple comparison with FUSE-based FS (direct flag not working, with cache)

Single NIC port full

1:1 seq read and write degraded a lot

CHAOS TEST

Distribution file system base FUSE and Lustre Use the same hardware

Parallel 8000 copy process in single client , copy 1.8M 4KiB files

The distribution FUSE-based FS only 17.73 files/s (Minio EC:0) 100% CPU

Lustre : 2182.46 files/s (no raid) , 50% CPU

123x performance lead was observed on Lustre

You can make the scores look good, but I can expose the lie



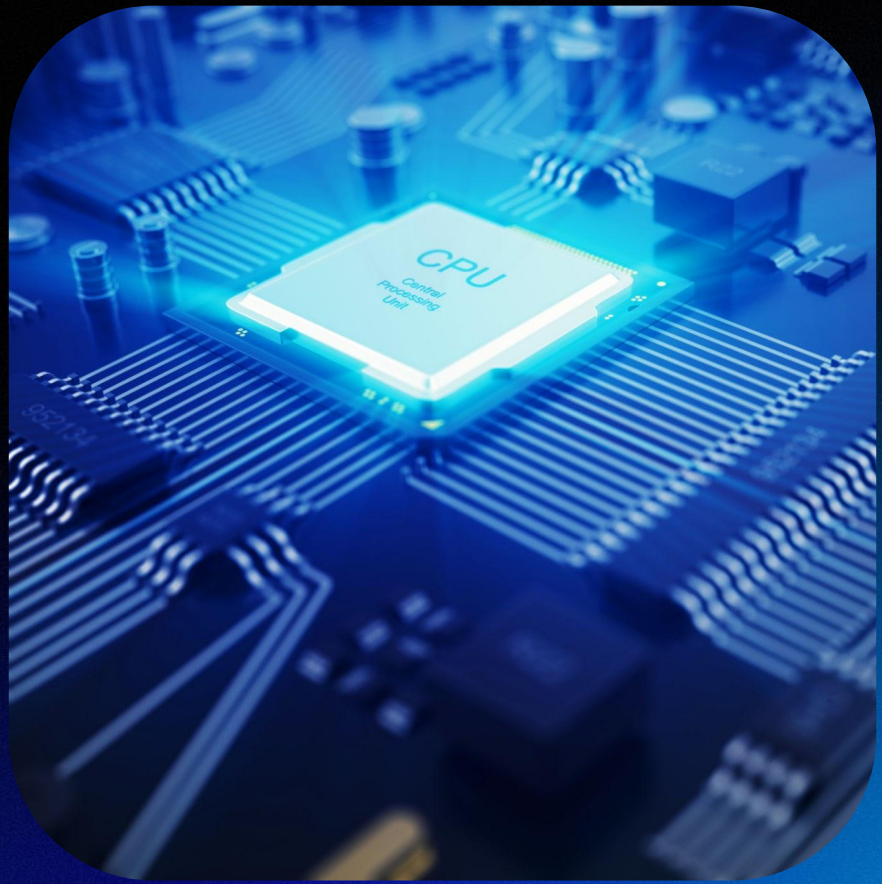


The ASIC is very fast and efficient at that class of tasks

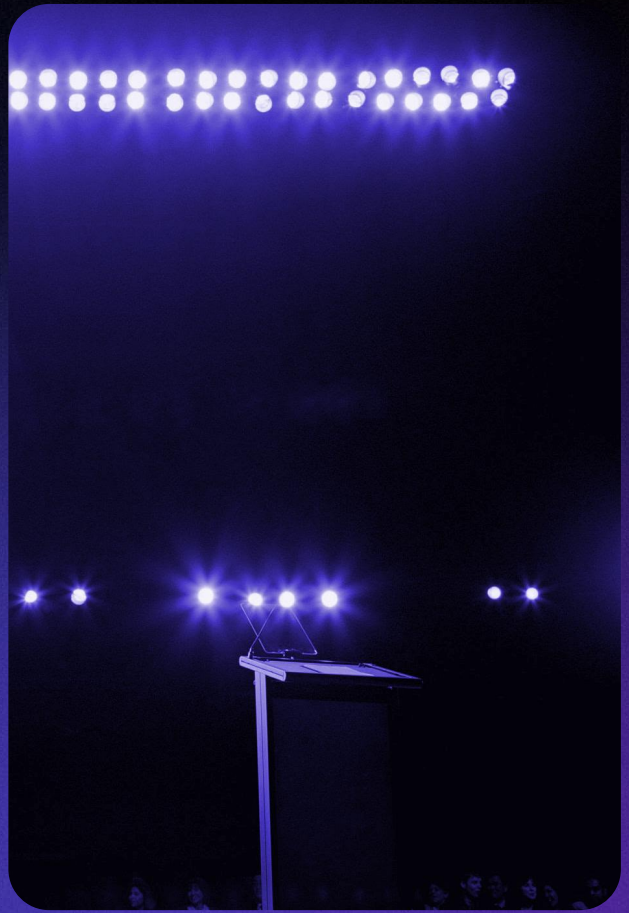
The CPU designed for many general tasks. It is flexible, but less efficient than an ASIC for a specific task

Similarly, high-performance Lustre and these FUSE-based FS are also like CPUs and ASICs

Swiss Army Knives VS Chef's Knives



In the vast darkness of the dataset, hot files are the shining stars



Production environments are not like tests where every file is accessed evenly.

The reality is different: there are more and more custom data index files, and high-frequency reads are especially noticeable

Basic

In uncomplicated scenarios, this detail is sufficient.

An in-depth analysis would require extensive logging to identify data hotspots

Here is an example

01

Lustre log

Enable debug, lctl dk

02

ZFS interface

/proc/spl/kstat/zfs/tank/reads

03

NIC throughput

Network monitor tools

Analysis directly (easy; no write script; lower overhead)

```
perf probe -m "/lib/modules/4.18.0-553.5.1.el8_lustre.x86_64/extra/lustre/fs/ofd.ko" --add "ofd_commitrw f_fid=oa->o_oi.oi_fid.f_seq:u64 f_oid=oa->o_oi.oi_fid.f_oid:u32 rwtype=cmd:u32"
```



```
ll_ost_io00_124 136662 [003] 2944725.609229: probe:ofd_commitrw: (ffffffffc160a830) f_fid=4295229440 f_oid=50005 rwtype=2
ll_ost_io00_123 136661 [001] 2944725.609864: probe:ofd_commitrw: (ffffffffc160a830) f_fid=4295229440 f_oid=50005 rwtype=1

./lustre/include/uapi/linux/lustre/lustre_idl.h:1379:
#define OBD_BRW_WRITE         0x02
./lustre/include/uapi/linux/lustre/lustre_idl.h:1378:
#define OBD_BRW_READ          0x01
```

```
awk 'BEGIN{a=4295229440;;b=rshift(a,16); c=and(b,0xffff);printf "OST index: 0x%x %d\n", c,c}'
OST index: 0x4  4

echo "O/0/d$((50005%32))/50005"
O/0/d21/50005

zdb -vv -bbb -O test_ost4/test_ost4 O/0/d21/50005
    Object  lvl   iblk   dblk  dsize  dnsize  lsize   %full  type
     51123    3   128K   128K   100G     512   100G  100.00  ZFS plain file
                                         192   bonus  System attributes
......
        trusted.fid =
\001\004\000\000\002\000\000\000\137\303\000\000\000\000\000\000\000\000\020\000\001\000\000\000\000\000\000\000\000
```

ZFS Analysis

```
echo
"\001\004\000\000\002\000\000\000\137\303\000\000\000\000\000\000\000\000\020\000\001\000\000\000\000\000\000\000\000\000\000\00
0\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000" | awk 'BEGIN{printf "[0x";} {gsub(/\\/, "\n", $0);
  for (i=8;i>=1;i--) {hex=sprintf("%02x", strtonum("0" $i)); printf hex };
  printf ":0x";
  for (i=12;i>=9;i--) {hex=sprintf("%02x", strtonum("0" $i)); printf hex };
  printf ":0x";
  for (i=16;i>=13;i--) {hex=sprintf("%02x", strtonum("0" $i)); printf hex };
print "]"}'
[0x0000000200000401:0x0000c35f:0x00000000]

lfs fid2path /mnt/lustre [0x0000000200000401:0x0000c35f:0x00000000]
 /mnt/lustre/zfs_4/100G-2.file
```

A challenge with debugging ZFS is that as I/O operations go deeper into the stack, they are aggregated, which obscures the link to the original dnode. If trace on TrueNAS system, analysis at the ZPL layer

Others

by IO type

perf probe -m "/lib/modules/4.18.0-553.5.1.el8_lustre.x86_64/extra/lustre/fs/ofd.ko" --add "ofd_sync_hdl f_fid=tsi->tsi_fid.f_seq:u64 f_oid=tsi->tsi_fid.f_oid:u32 f_ver=tsi->tsi_fid.f_ver:u32"

ll_ost00_019 420891 [001] 2949072.963700: probe:ofd_sync_hdl: (ffffffffc15ea400) f_fid=4295032832 f_oid=10926 f_ver=0
ll_ost00_019 420891 [001] 2949072.979283: probe:ofd_sync_hdl: (ffffffffc15ea400) f_fid=4295032832 f_oid=10926 f_ver=0
ll_ost00_019 420891 [001] 2949072.994232: probe:ofd_sync_hdl: (ffffffffc15ea400) f_fid=4295032832 f_oid=10926 f_ver=0
ll_ost00_019 420891 [001] 2949073.009241: probe:ofd_sync_hdl: (ffffffffc15ea400) f_fid=4295032832 f_oid=10926 f_ver=0

Get the zfs dnode directly, trace files generator

perf probe -m zfs "dnode_hold dnode_id=dn.dn_object:u64"

ll_ost00_008 189678 [017] 244109.194807: probe:dnode_hold: (ffffffffc141bf50) dnode_id=113468
ll_ost00_008 189678 [017] 244109.194810: probe:dnode_hold: (ffffffffc141bf50) dnode_id=113468
ll_ost00_008 189678 [017] 244109.195290: probe:dnode_hold: (ffffffffc141bf50) dnode_id=113469
ll_ost00_008 189678 [017] 244109.195291: probe:dnode_hold: (ffffffffc141bf50) dnode_id=113469

lustre/include/uapi/linux/lustre/lustre_fid.h
lustre/include/uapi/linux/lustre/lustre_user.h

To measure the latency, we trace the entry and exit times of the system call to get the high latency file info.

Ldiskfs Analysis

For instance, blktrace output might that many I/O block on this inode

```
blktrace -d /dev/sdf -a read -o - | blkparse -i -

8,80  16    1556    3.216571666  3556  A   R 1529444352 + 4096 <- (8,81)
1529442304


debugfs -R "icheck $((1529444352/8))" /dev/sdf1
Block    Inode number
191180544    3678857

debugfs -R "ncheck 3678857"  /dev/sdf1
Inode    Pathname
3678857    /O/0/d8/6246888


debugfs -R "stat /O/0/d8/6246888"  /dev/sdf1
Inode: 3678857   Type: regular    Mode:  0666   Flags: 0x80000
.....
Extended attributes:
  lma: fid=[0x100030000:0x5f51e8:0x0] compat=8 incompat=0
  fid: parent=[0x2000004ca:0xfaf6:0x0] stripe=0 stripe_size=1048576
stripe_count=1

lfs fid2path /mnt/lustre [0x2000004ca:0xfaf6:0x0]
/mnt/lustre/test/test_33
```

MDT troubleshooting tool

- Lustre/scripts/remove_updatelog
  - This is emergency tool to cleanup changelogs in server if changelog, records cannot be removed by regular means, e.g. due to llog corruptions
- Lustre/scripts/remove_changelog
  - This is emergency tool to cleanup updatelogs in server if llog records cannot be removed by regular means, e.g. due to llog corruptions

Script provides a guided process that mitigates risks and eliminates the need for manual deletion

谢谢!
Thank you!