LFSCK

High Performance Data Division

Alexey Zhuravlev

April 17, 2013

LFSCK: before

- lfsck databases created via e2fsck
  - not suitable for ZFS, btrfs, …
  - very slow due to large database size
  - need network storage or database copy
- complex to keep in sync with Lustre* code
- not very useful for large file systems

LFSCK: online Lustre File System Checker

- The need in LFSCK is driven by:
  - Lustre isn't free of bugs, some may lead to on-disk inconsistency
  - Same for hardware
  - New features may require new on-disk format
    - Backup/restore isn't an option in many cases
- The checks should be running online:
  - The clusters grow quickly: in files, byte and in nodes
  - The checks can take a lot of time, even being run exclusively
    - Lfsck can take weeks on large file system
  - Waiting for an error is not very wise
    - It's better to rescue data proactively, so they're ready for the users
  - Should not affect visible performance much (unless required)

Changes on-disk since 1.8: OI – Object Index

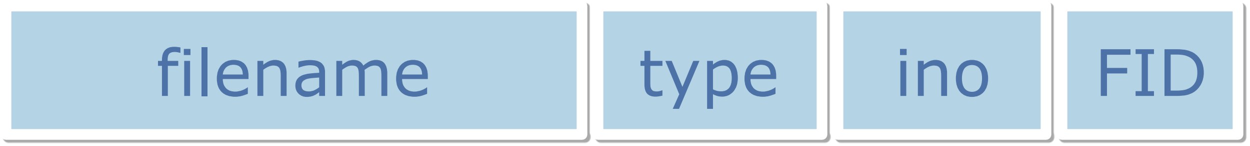
- Maps FIDs to local IDs (inode, dnode)
  - FID is Lustre Unique File ID
- Every specific OSD implements own OI internally
  - Every OSD hides details of underlying filesystem (ldiskfs, ZFS, btrfs)
- Correct OI is vital for Lustre to behave properly
- 2.X supports old inode/generation IDs in *compatibility* mode, but the functionality is limited
  - Subset of FIDs (IGIFs) is reserved to map FIDs to inodes directly
  - Can be used only with existing setups
  - IGIF can't be created with 2.X

OI: what and how to fix

- Missing FIDs and OI in 1.8 in case of upgrade to 2.x
  - To enable new features like path2fid
- Stale OI in case of file level backup/restore on MDS
  - Restore does not put files in the same inodes
- Errors introduced by bugs/issues with software/hardware
- Since 2.x every object has LMA attribute
  - Storing object's FID
- Collection of LMA is reverse OI
  - Can be used to verify and fix OI

Changes on-disk since 1.8: FIDs in directories

- We do store local object IDs (inode, dnode) in directory entries
  - To preserve on-disk compatibility with local file system
  - E.g. you can still mount with –t ldiskfs, use fsck.extN, etc
- READDIR returns <name; object id; type> triples
- We could get FID from LMA stored in object
  - But this is extra seek and IO to get local object – performance penalty
- We need both FID and local ID in the entries
  - To preserve compatibility and performance

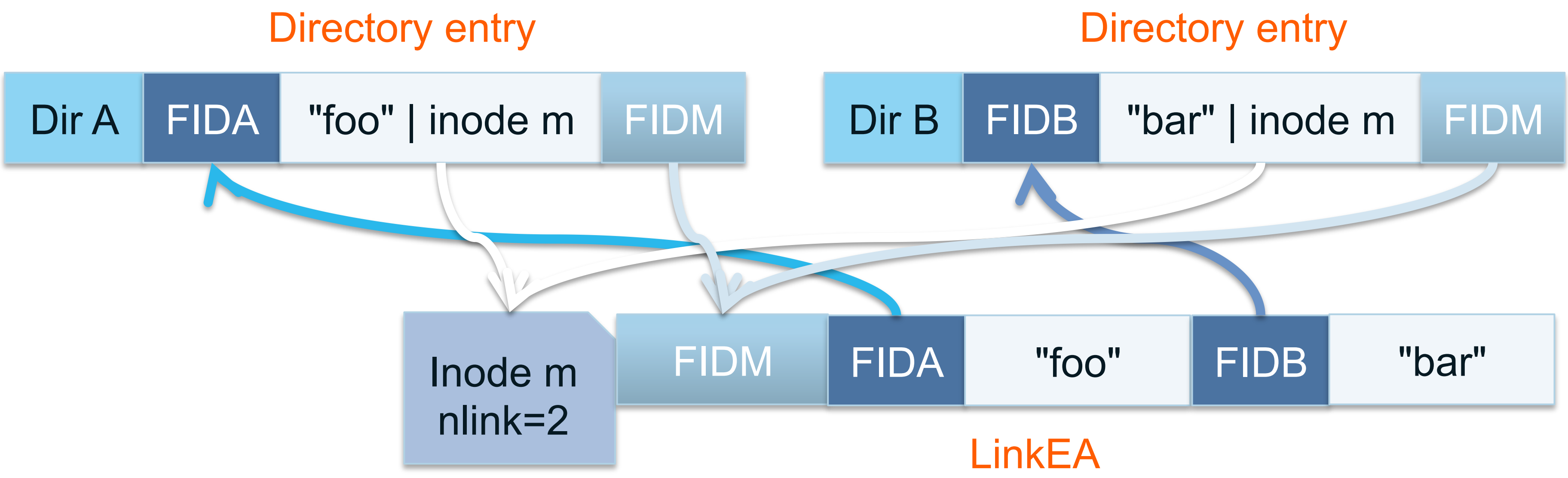


filename	type	ino	FID

FIDs in directories: use cases

- 1.8 file systems do not have FIDs in directory entries
- Need to upgrade the entries if upgraded to 2.X
- FIDs in directories are lost after file level backup/restore
- Better to be done online
- On the access from the clients

Changes on-disk since 1.8: LinkEA



- Extended attribute for every file/directory
- Helps to find path name to object with specific FID
- Introduced in 2.0 with limits, mandatory since 2.4
- Requires stable FIDs

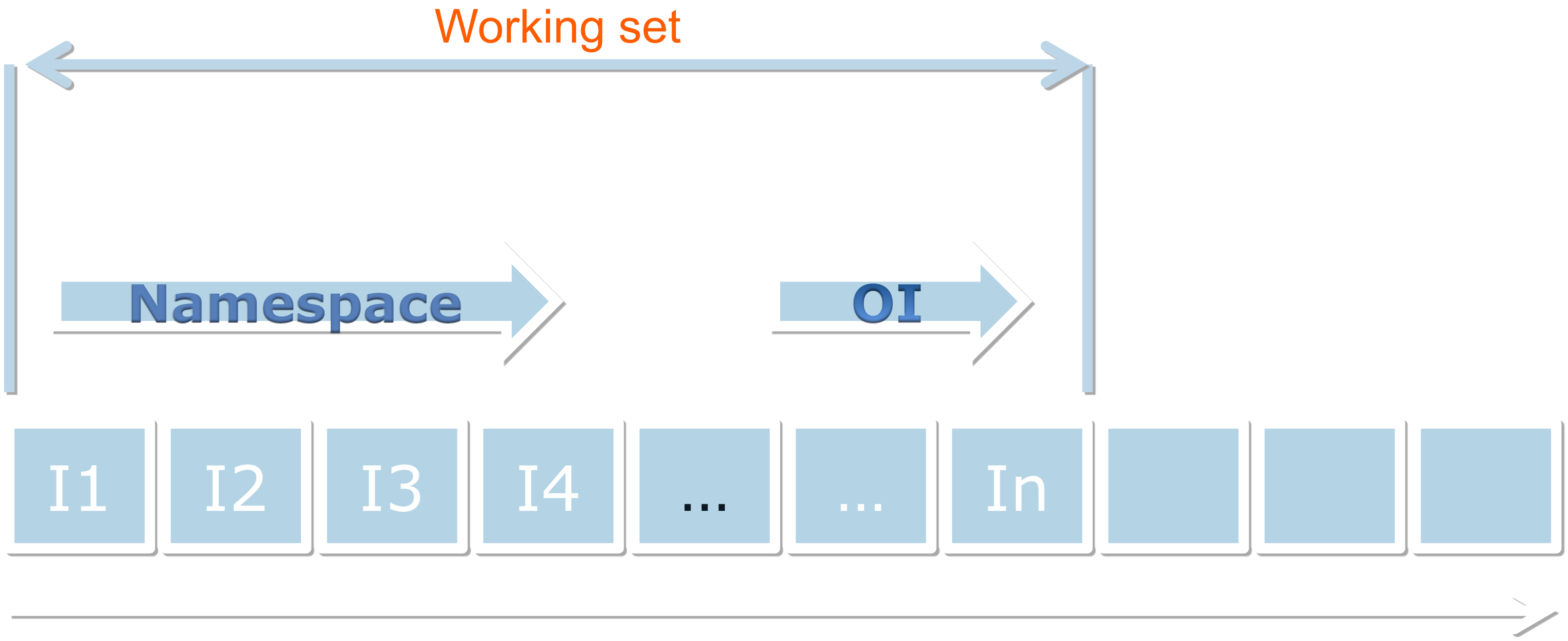
LinkEA: what and how to fix

- Missing LinkEA after upgrade from 1.8
- Software/hardware issues
- Scan a directory, verify every name is presented in LinkEA
- Trust visible namespace over LinkEA
- Files with multiple names (hardlinks) need special handling
  - Build an index listing objects with many names
  - Remove object from the list once all the referenced verified
  - One more scan for the objects remaining in the list
  - Supposed to be small usually – the case is relatively rare

Lots of different checks: fits together

- Few totally independent scanners could be implemented
  - One to check OI
  - Another to check LinkEA
  - Yet another to repair/rebuild FIDs in directories
- Not very efficient from I/O throughput perspective
- Implemented like a pipe
  - Two independent scanners
  - Same set of objects
  - Data is fetched once

Piped LFSCK Processing



Working set

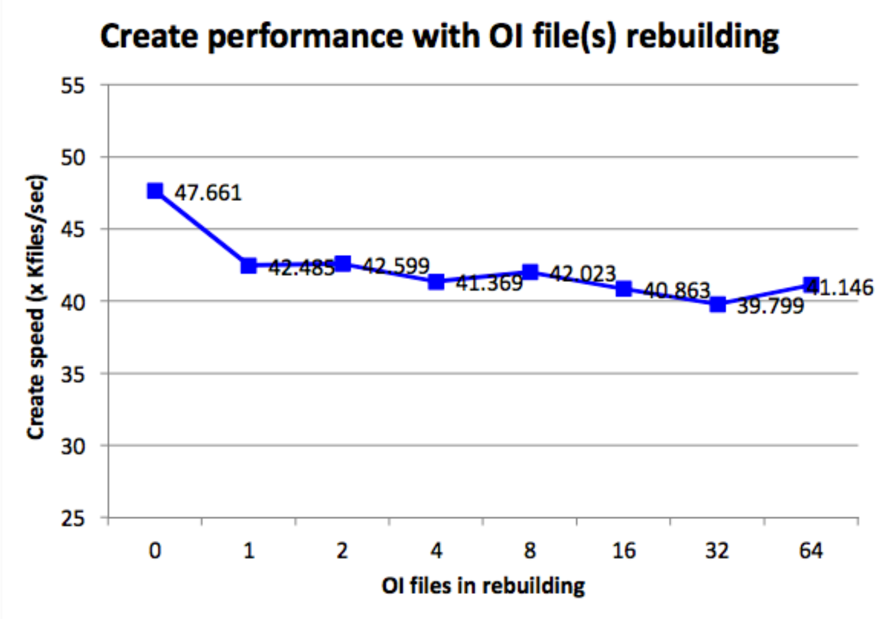
Namespace

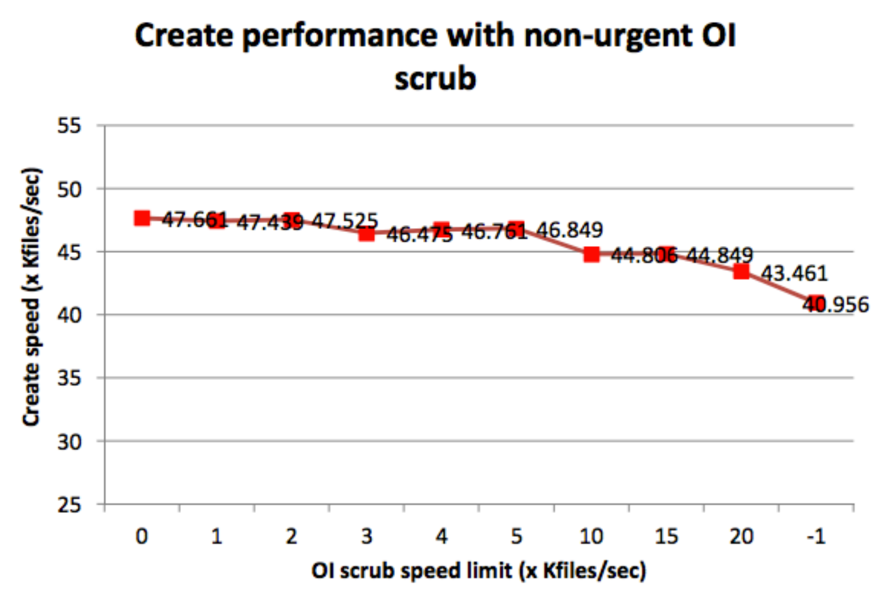
OI

I1 | I2 | I3 | I4 | ... | ... | In

Inode table

Performance





Next step: LFSCK 2

- Consistency between MDS and OST
  - Missing OST objects
  - Lost OST objects
  - OST objects referenced by few files
- The architecture and the design are ready

Yes another step: LFSCK 3

- New set of inconsistencies with DNE
  - Name pointing onto non-existing objects
  - Lost files and directories
- Vital to DNE phase 2
  - The plan B for the cases we might miss in the design
  - Good option for setups where metadata performance is very important and reboots are rare
- Online
- The scope can be restricted to set of objects
  - No need to scan through all objects: just one affected by recent distributed operations

LFSCK and ZFS

- ZFS is almost first class citizen backend
  - Declared OSS on ZFS support in 2.3, Fully functional in 2.4
- No need to upgrade on-disk format
  - Features like LinkEA, FID-in-dirent supported from the beginning
- Less need in OI scrubber
  - ZFS has own mechanism to maintain consistency
- Object Iterator is still needed
  - To implement LFSCK 2 and LFSCK 3 – distributed cases
- Provides the bookmark mechanism to walk through a tree
  - We need to export it with OSD API

intel®