SEAGATE

Scaling LDISKFS for the future. Again

Artem Blagodarenko

LUG 2017
Bloomington, Indiana

LDISKFS still grows

As drive size increases

...8TB -> 10TB -> 12TB

The maximum backend storage size increases

...16TB -> 500TB

LDISKFS quickly exceeded the original design!

The summary of previous work

✓ Done

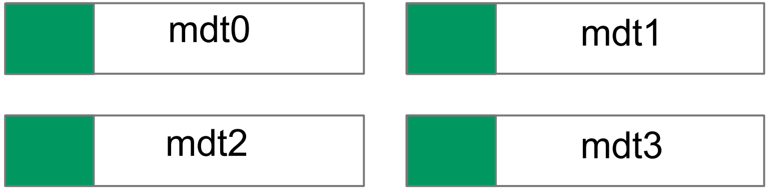
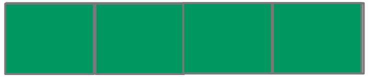
- *code review*
- *testing suite*
- *patches with fixes*
- *move LDISKFS size limit to* ***256TB*** *(**LU-7592**).*

! Problems

- inodes count over UINT32_MAX
- large memory blocks allocation
- solution for large directories

Inode count limit (LU-1365)

Example: a customer requires 16 billions of inodes on MDS

Only 4 billions of inodes on one MDT	Unfortunately we can not make 16 billions inodes on one MDT because of LDISKFS limitation
	We can use 4 MDTs with **DNE** but MDT's space is not completely used
 16 billions	>4 billions inodes on LDISKFS

Inode count limit. Additional fields for ext4_dir_entry

Offset	Size	Name	Description
0x0	__le32	inode	Inode number
0x4	__le16	rec_len	Length of this directory entry
0x6	__u8	name_len	Length of the file name
0x7	__u8	file_type	File type (0x0F), Dirdata (0xF0)
0x8	__u8	lufid_len	OST fid length
0x9	N	fid	EXT4_DIRENT_LUFID
0x8 + N	__u8	hi_inode_len	length, always 4
0x8 + N + 1	__le64	hi_inode	EXT4_DIRENT_INODE

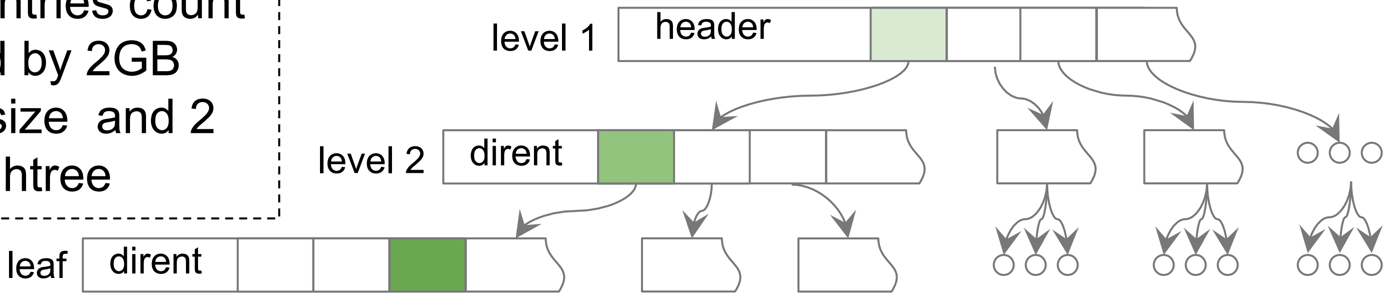
dirdata pros and cons

- + less space for 64-bit inodes
- + smaller dirents for 32-bit inodes
- + more 32-bit dirents in leaf block
- + backwards compatible with existing directories
- + doesn’t require full update

- − not obvious
- − requires some extra code

Large directory (LU-1365)

LDISKFS entries count is limited by 2GB directory size and 2 level htree



$$\text{index 1} = \frac{\text{block size - header size}}{\text{index size}}$$

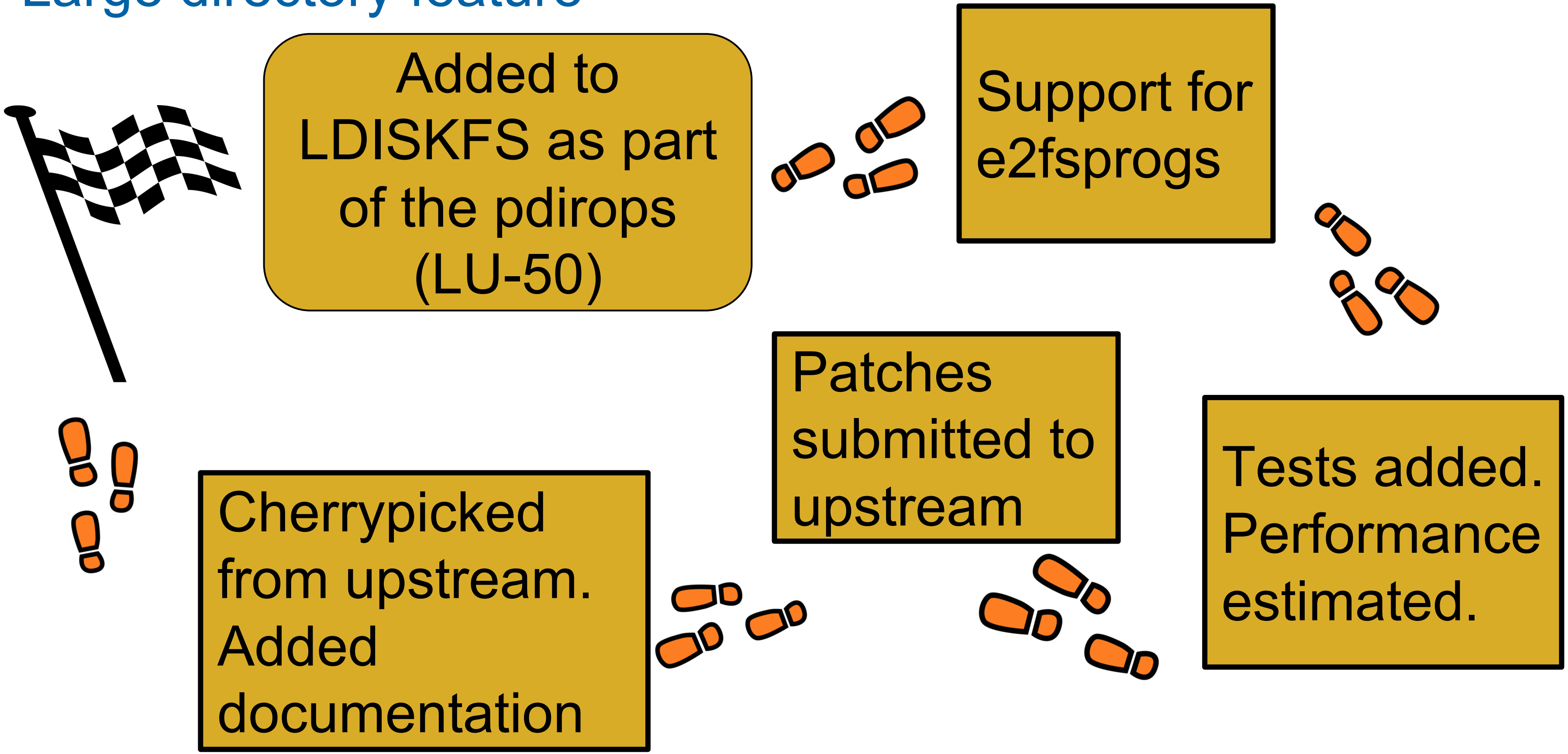
$$\text{dirent in leaf} = \frac{\text{block size}}{\text{name length + dirent size}}$$

$$\text{index 2} = \frac{\text{block size - header size}}{\text{index size}}$$

entries in directory = index 1 * index 2 * dirent in leaf

Single directory capacity depends from names size and for average file system is ~10 millions of entries

Large directory feature



Cherrypicked from upstream. Added documentation

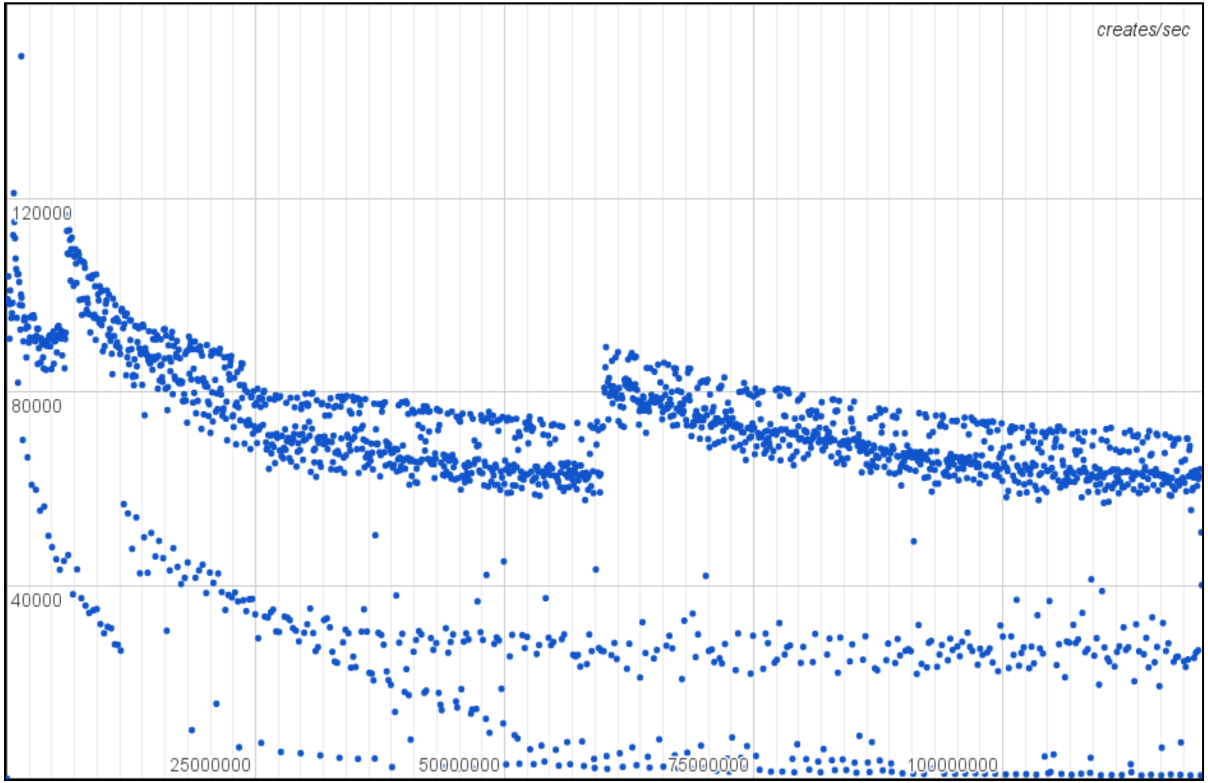
Patches submitted to upstream

Tests added. Performance estimated.

Support for e2fsprogs

Added to LDISKFS as part of the pdirops (LU-50)

testing large_dir



config_sanity.sh 101
"Adding large_dir with 3-level htree"

config_sanity.sh 102
"Adding large_dir with over 2GB directory"

- 120M
- ldiskfs only
- hard links
- createmany utility

Challenges

- On large mdt targets before 64-bit inode counter patch is landed inode number can be > 4 billions. In this case formatting is finished with error. Adjusted automatically (LU-9501).
- Large memory structures. Code inspection.
- Group blocks count exceeds EXT4 design
- LU-8444 ldiskfs_xattr_inode_iget: error while reading EA inode -2147483347" on large MDT volumes with large_xattr feature enabled (test added, LU-8444)

Group blocks count problem

- all block group descriptors copies are kept in the first block group
- Given the default 128MiB(2^27 bytes) block group size and 64-byte group descriptors, ext4 can have at most 2^27/64 = **2^21** block groups
- This limits the entire filesystem size to 2^21 ∗ 2^27 = **2^48** bytes or **256 TiB**

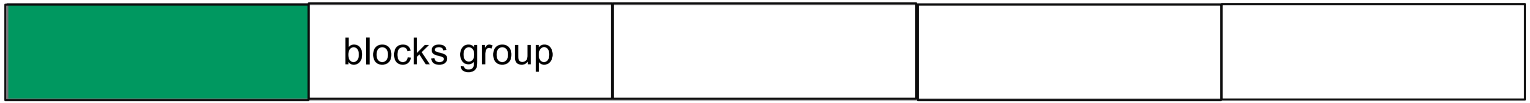
Two solutions:

✓ meta_bg

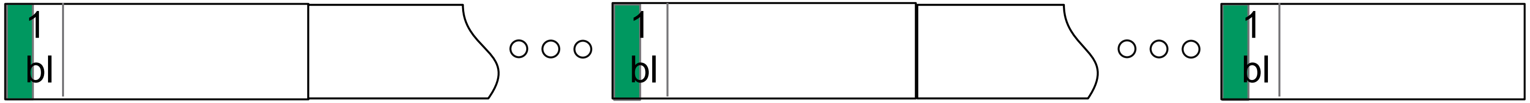
✓ bigalloc

We get **meta_bg** feature as solution

Meta_bg feature



Meta_bg allowing support for a 512PiB filesystem



meta_bg is the obvious way to solve the trouble with the group descriptors. They will not be able to fit into the first group after it grows beyond some count of blocks (because partition is too large). **meta_bg** solve this problem, so we can have as many block groups as we need.

Meta_bg. Patches

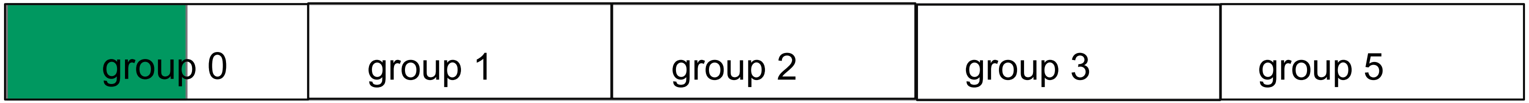
- LU-9501 "libext2fs: automatically enable meta_bg to avoid filling up BG 0"
- LU-9160 libext2: readahead for meta_bg
- ldiskfs: preload block groups. landed to EXT4 and e2fsprogs
- LU-8976 Apply patch "libext2fs: fix maximum bg overhead calculation with meta_bg enabled"
- LU-8443 utils: exclude "resize" parameter with meta_bg option

Loading metadata during mount (LU-9160)

With enabled meta_bg option block group descriptors reading IO is not sequential and requires optimization.
Example:

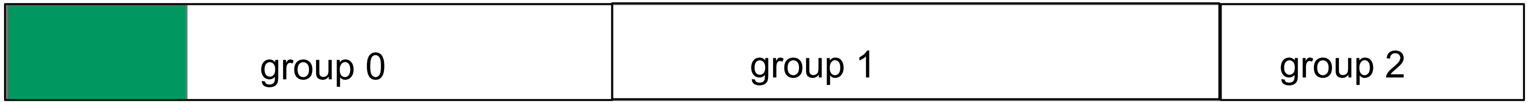
- There are ~37k of random IOs with meta_bg option on 300T target.
- Debugfs requires 20 minutes to be started.
- Enabling readahead for group blocks metadata save time dramatically. Only 12s to start. (landed to EXT4)

Bigalloc



group 0 group 1 group 2 group 3 group 5

The administrator can set a block cluster size

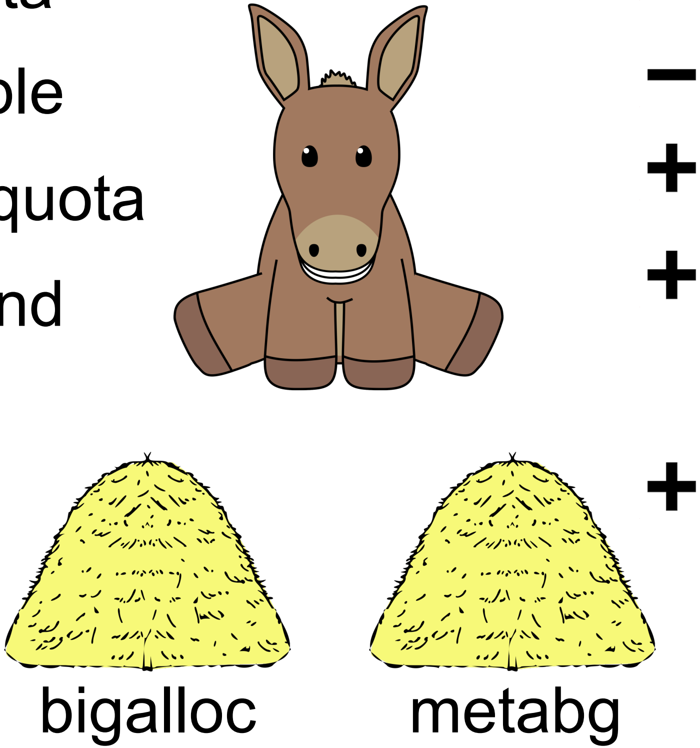


group 0 group 1 group 2

Bigalloc feature decreases the needed number of blocks groups, because “block” (called cluster) became bigger (for example 64k against the 4k).

Bigalloc vs meta_bg

- **+** Less metadata
- **–** Looks unstable (issues with quota and links found during tests)
- **–** not good for small files



- **–** more metadata
- **–** memory usage
- **+** passed testing
- **+** good for small files
- **+** can be applied to existing systems

Testing

- *Mount the 256 TB+ device as ldiskfs to ensure lustre/kernel supports huge file systems*
- *Run e2fsprogs utilities to ensure 256 TB+ support*
- *Running modified xfstest for stress testing*

- *Run llverfs and lldevfs to ensure that the kernel can perform operations on the device without any errors*
- *Setup OST on this device to ensure Lustre can handle huge devices and run Lustre testsuite*

Components To Be Tested		
e2fsprogs	*ldiskfs*	*Lustre*

Results

To address concerns regarding these issues Seagate has developed an **open source code review** and **updated testing suite.** The suite has successfully verified new patches that improved performance, resulting in open source **upstreamed patches** increasing the ldiskfs size limit to **512TB** (**LU-8974**).

This work allows customers to **have fewer, larger OSTs** resulting in decreased resource requirements on clients customers and allows customers to deploy have denser storage.

Current work

- large dir landing
- 64 inode inode pointer in progress
- bigalloc testing and adapting to Lustre FS

Current work is focused on researching possible scaling problems and providing solutions for extending the limit above 512TB.

Acknowledgments

Thanks to

Alexey Lyashkov (Seagate)
Elena Gryaznova (Seagate)
Andreas Dilger (Intel)

SEAGATE

Thank you!