

LDISKFS BLOCK ALLOCATOR AND AGED FILE SYSTEM

Artem Blagodarenko
Paris, LAD 2019



CRAY[®]

Symptoms: user point of view

File system running a job which usually takes 12 seconds to complete, now taking up to 12 minutes

- Fresh file system



High write performance



High read performance



No fragmentation

- Aged file system



Extremely low write performance



Same read performance



New data is fragmented

A lot of free blocks
736.29 GB/s

**6000% decrease
in performance**

76% – 81% filled
13.37 GB/s

Symptoms: developer point of view

```
xfffffffafa14468bd <ldiskfs_mb_regular_allocator+589>: mov -0x6c(%rbp),%edx
0xffffffffffa14468c0 <ldiskfs_mb_regular_allocator+592>: mov %ebx,%esi
0xffffffffffa14468c2 <ldiskfs_mb_regular_allocator+594>: mov %r12,%rdi
0xffffffffffa14468c5 <ldiskfs_mb_regular_allocator+597>: callq 0xffffffffffa1444c70
<ldiskfs_mb_good_group>
```

```
RDX: 0000000000000001 - cr
ESI 0000000000000008 - group
RDI: ffff881f52e24000 - allocation
context
```

**allocator started
scanning from
group 753402 and
now processing
group 8**

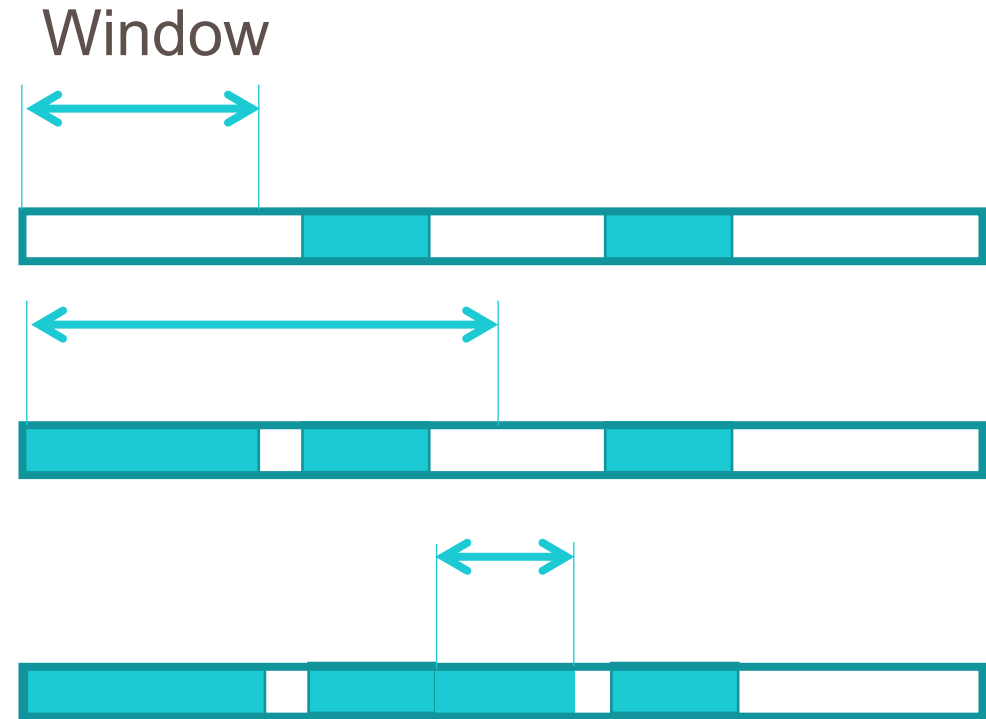
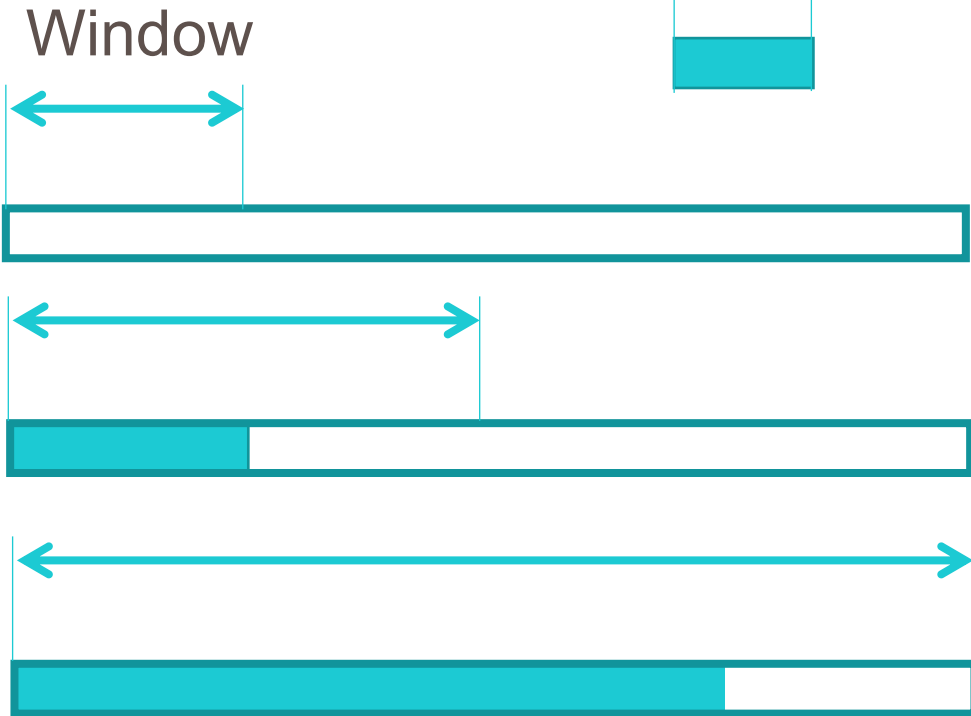
```
crash> struct ldiskfs_allocation_context
ffff881f0ba1b300
struct ldiskfs_allocation_context {
ac_g_ex = {
fe_logical = 370218,
fe_start = 13604,
fe_group = 753396,
fe_len = 6614
...
ac_groups_scanned = 0,
ac_found = 0,
```


Problem

unfragmented

requested

fragmented



Allocator processes whole disk trying to find large continuous range of blocks. Disks become larger, the problem becomes visible.



Buddy allocator

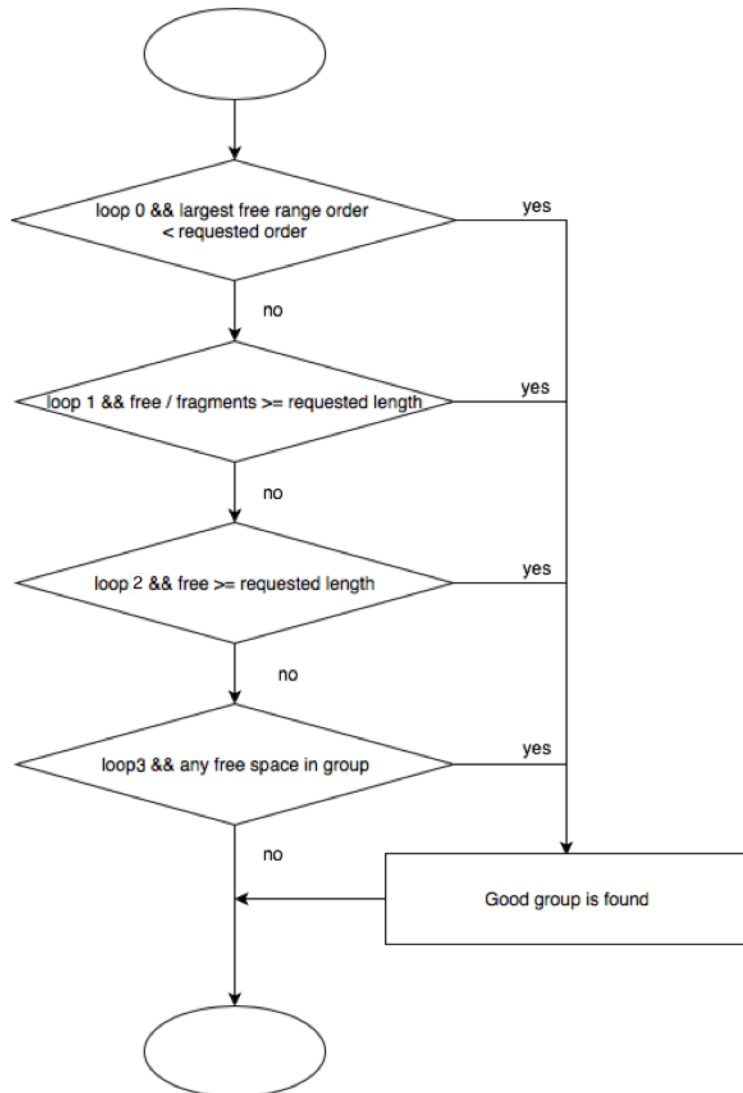
Conditions:

- File size is bigger than **s_mb_stream_request** or group preallocation can't find required blocks
- Required blocks are not found in inode preallocation list
- Required blocks are not found in locality group pre-alloc space

Actions:

- **Normalizing** the request for size and alignment
- Pre-allocate the blocks
- Place the pre-allocated blocks into the pre-allocated block space or inode pre-allocated space (this decision is based on **EXT4_MB_HINT_GROUP_ALL** OC flag)

Buddy allocator loops



4 loops across all groups of filesystem

- cr=0, try to allocate 2 bytes
- cr=1, average free range has required size
- cr=2 Group has enough data
- cr=3 use any free data

Based on the file size requested (offset + requested size) block count is rounded to the nearest large block range e.g.: (16K, 32K, 64K, 128K, 256K, 512K, 1M, 2M, 4M, 8M, etc. – **preallocation table**)

Tunable and statistic

<code>mb_min_to_scan</code>	When good group (<code>is_group_good</code>) is found, the allocator will start to scan extents . <code>mb_min_to_scan</code> is extents count that must be scanned before the allocator decides on a possible group, <code>mb_max_to_scan</code> is the extents count after the allocator search is interrupted
<code>mb_max_to_scan</code>	
<code>s_bal_reqs</code>	number of requests
<code>s_bal_allocated</code>	found target chunk
<code>s_bal_ex_scanned</code>	how much extents scanned
<code>s_bal_goals</code>	block is allocated from goal
<code>s_bal_breaks</code>	allocator was interrupted (because of <code>s_mb_max_to_scan</code>)
<code>s_bal_2orders</code>	2^orders hits
Statistic is shown on fs unmount (<code>ext4_mb_release()</code>) if <code>mb_stats</code> is set.	

Solutions



Adjust preallocation table to limit preallocation window grow



Preallocation table:

16 32 64 128 256 512 1024 2048

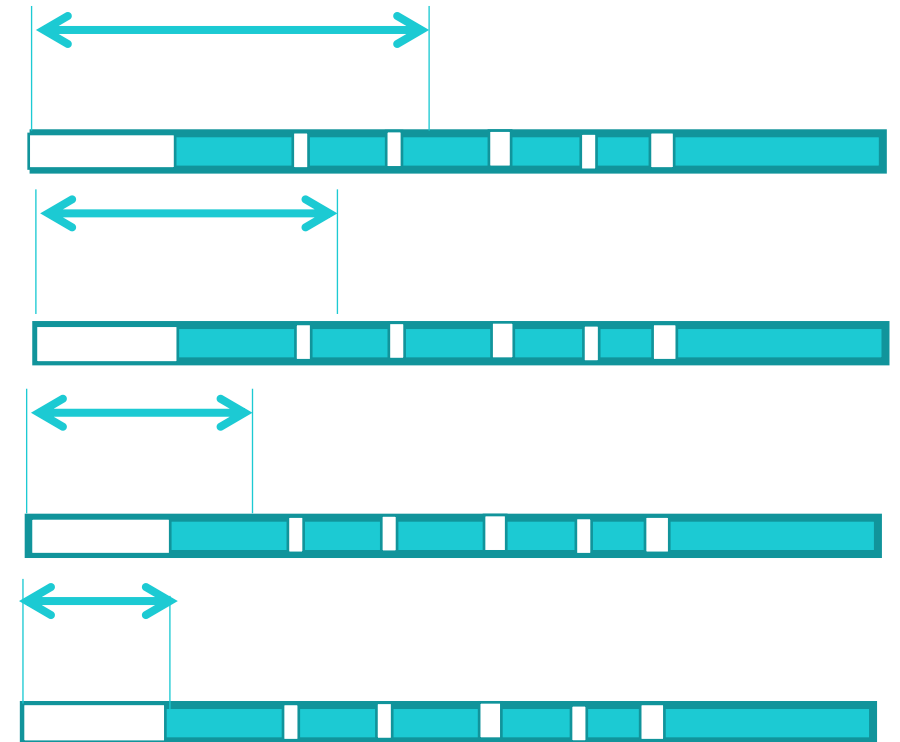
Exclude all impossible values

Modified preallocation table:

16 32 64 128 256 512



Force to skip "useless" allocator loops



Preallocation table solution

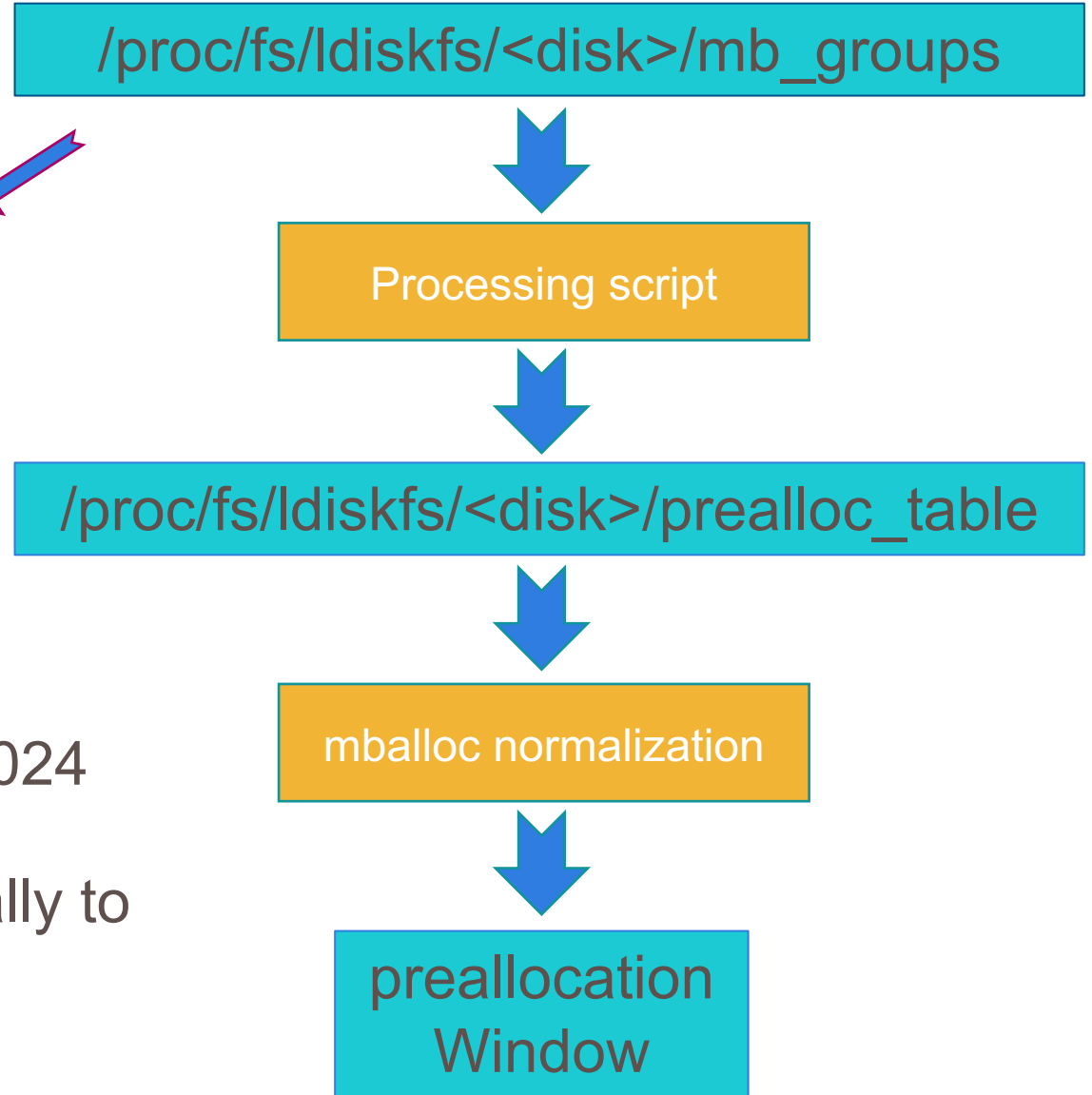
[2 ⁰	...	2 ¹³]
[0 1 1 0 0 1 0 1 1 1 1 0 0 0]		
[1 0 1 1 2 2 1 1 3 1 1 1 0 0]		
[0 0 0 0 0 1 0 0 1 0 0 1 0 0]		
1 2 4 8 16 32 64 128 256 512 1024 2048 4096		

1 2 4 8 16 32 64 128 256 512 1024 →

Offset + requested size = 3000

Normalized request 4096 but is limited to 1024

prealloc_table should be adjusted periodically to reflect current state



Before table adjustment

1. Set too large preallocation table and estimate write speed

```
echo "256 512 1024 2048 4096 8192 16384"
```

```
dd if=/dev/zero of=/mnt/mntpnt/O/foofile bs=1048576 count=1024 conv=fsync
```

1073741824 bytes (1.1 GB) copied, **11.2427** s, **95.5** MB/s

mballoc: 262144 blocks 153 reqs (137 success)

mballoc: **2046** extents **scanned**, 127 goal hits, 1 2^N hits, **10 breaks**, 0 lost

After table adjustment

2. Adjust preallocation table based on mb_groups output

```
cat /proc/fs/ldiskfs/loop1/mb_groups > $TMP/table.dat
```

```
sh build_prealloc.sh $TMP/table.dat > $TMP/prealloc.txt
```

```
cat $TMP/prealloc.txt > /proc/fs/ldiskfs/loop1/prealloc_table
```

```
dd if=/dev/zero of=/mnt/fs2ost/O/foofile bs=1048576 count=1024 conv=fsync
```

1073741824 bytes (1.1 GB) copied, **9.22825** s, **116** MB/s

mballoc: 262143 blocks 243 reqs (240 success)

mballoc: **141** extents **scanned**, 113 goal hits, 129 2^N hits, **0 breaks**, 0 lost

Issues



Lustre FS changes ext4 sources

https://github.com/tweag/lustre/blob/master/ldiskfs/kernel_patches/patches/rhel7/ext4-prealloc.patch

The reason - add striping.

Simplified seq file is used - only show() method (mb_prealloc_table_seq_show)

and ext4_mb_prealloc_table_proc_write - set table. The patch changes normalization algorithm.



LU-12335 ldiskfs: fixed size preallocation table. Preallocation table read/write code is racy. There is a possibility of accessing memory outside of allocated table. Make preallocation table fixed size. Array with 64 long int values are enough for any configuration and don't need much memory. With such array races are not possible.

Loops Skipping Solution

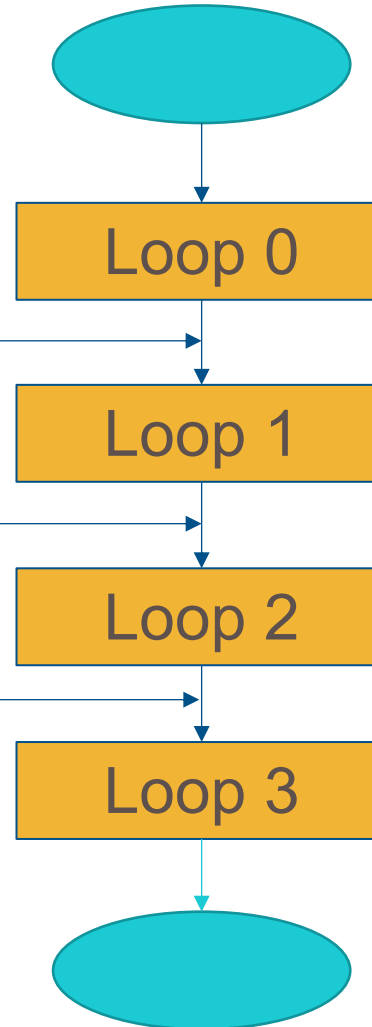
based on FS
condition

Start here if 75%
of disk is filled

Start here if 85%
of disk is filled

Start here if 95%
of disk is filled

force to skip
useless loops



Usage



```
echo "75" > /sys/fs/ldiskfs/loop1/mb_c1_threshold  
echo "85" > /sys/fs/ldiskfs/loop1/mb_c2_threshold  
echo "95" > /sys/fs/ldiskfs/loop1/mb_c3_threshold
```

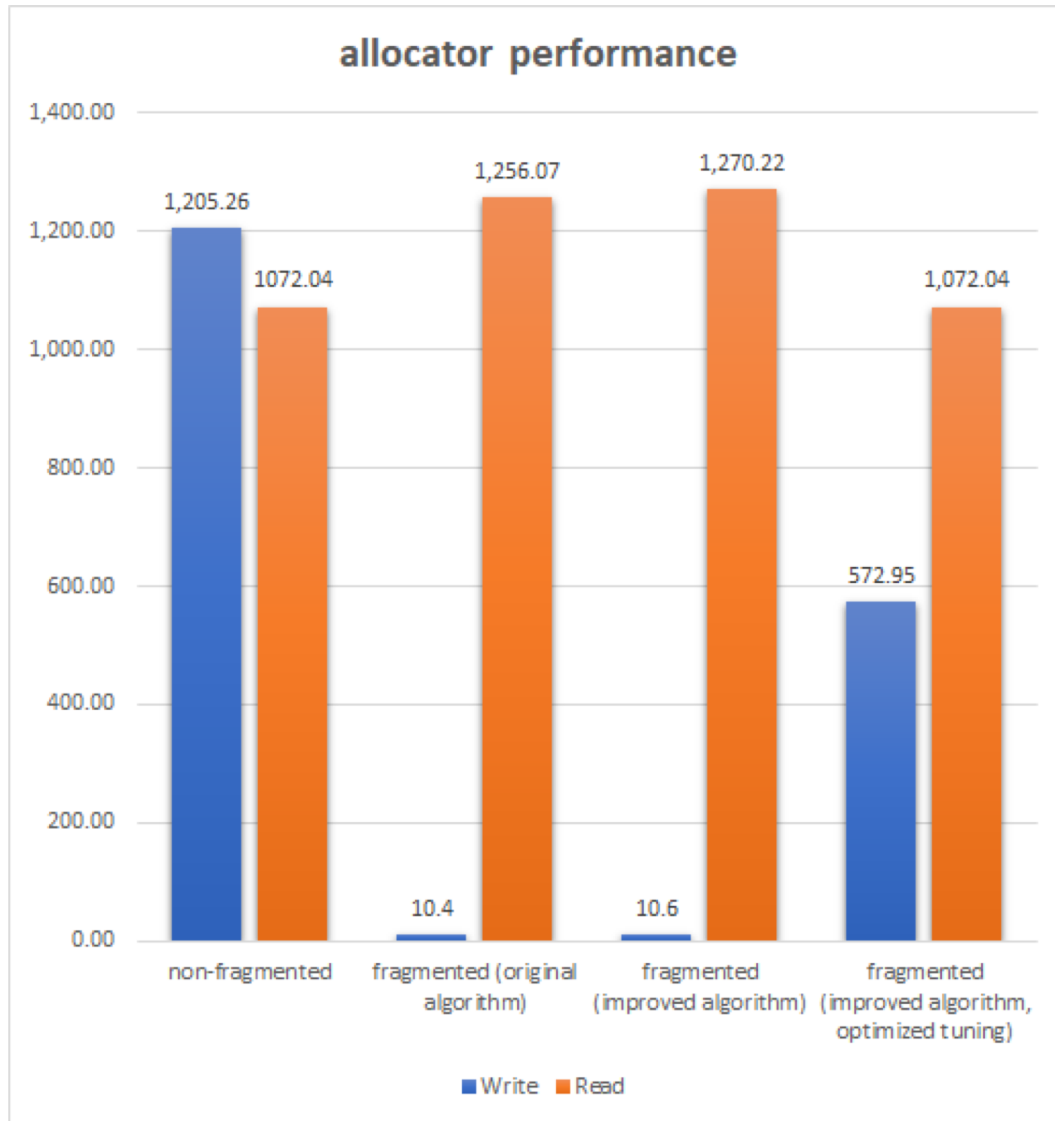


New strings are added to statistics
output (if mb_stats enabled)

```
mballoc: (349, 796, 0) useless c(0,1,2) loops
```


```
mballoc: (0, 0, 0) skipped c(0,1,2) loops
```


Testing

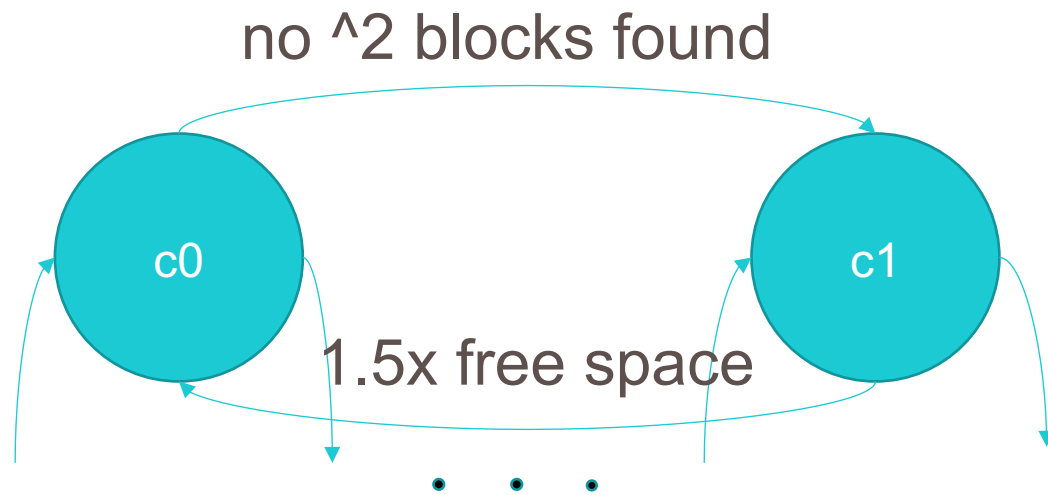



- Testing was made on ~100TB LDISKFS OST target:
- dd on non-fragmented block device shows **1.2 Gb/s** write and **1.0 GB/s** read
- Fragment partition with pattern: 50 free blocks – 50 occupied blocks using patched debugfs
- **99.24%** write performance reduction, reading not changed
- Set skip c0 loop if disk fragmented > 50%
- Write performance is not changed
- Set skip c0 and c1 if disk fragmented > 50%
- Write performance **557 MB/s** – **55%** from original


Heuristics

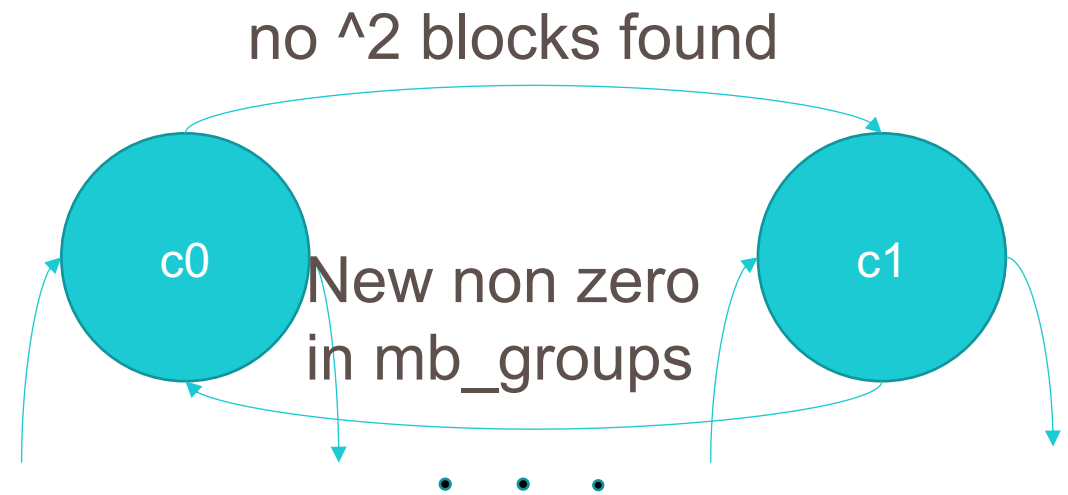
 Disable loops based on result of processing

 Reenable based on free space



 Disable loops based on result of processing

 Reenable based on largest free blocks range



Other possible solutions



big_alloc ext4 feature: blocks are grouped to clusters – fewer metadata, fewer groups



buddy allocator has several loops over all groups to obtain good one. But steps take similar actions so can be merged into single step



track maximal contiguous block region and stop extending allocation window if it is larger than maximum

Summary



LU-12103 patch is being landed, and it adds:

- Additional allocator statistics
- Options to enable heuristic for allocator loops skipping
- Heuristic, based on free space information



Ext4 is ready to accept the patch if no manual options are needed by user.



Preallocation table adjusting script is being tested on real cluster environment.

SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.



QUESTIONS?



artem.blagodarenko@gmail.com



[blagodarenko](https://twitter.com/blagodarenko)



[linkedin.com/in/artem-blagodarenko-a68b892b/](https://www.linkedin.com/in/artem-blagodarenko-a68b892b/)



[cray.com](https://www.cray.com)



[@cray_inc](https://twitter.com/cray_inc)



[linkedin.com/company/cray-inc/](https://www.linkedin.com/company/cray-inc/)

