Oak Ridge National Laboratory

Advanced Lustre File Layouts

Rick Mohr
Senior HPC Systems Engineer
Oak Ridge National Laboratory

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Overview

This tutorial will cover…

- Normal file layouts
- Progressive File Layouts (PFL)
- Data on MDT (DoM)
- File Level Redundancy (FLR)
- Self-Extending Layouts (SEL)

… but it will NOT cover

- Directory striping
- Choosing file layouts to optimize I/O performance

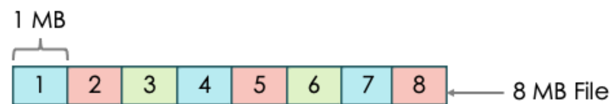
OAK RIDGE National Laboratory

This tutorial will cover some of the advanced file layouts available in Lustre. (NOTE: This will not cover directory striping.)

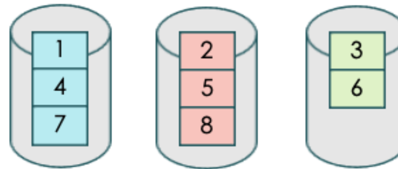
If we run out of time, slides will be posted so that the examples can be reviewed.

Normal File Layouts

- File is split into chunks and distributed across selected Object Storage Targets (OSTs) in round-robin fashion
- Splitting is controlled by two main parameters
  - Stripe count
  - Stripe size



Example:
stripe_count = 3
stripe_size = 1 MB



3 OAK RIDGE National Laboratory

This layout is usually the most familiar to users.

They will typically adjust the stripe count but may also adjust the stripe size if they are familiar with their I/O patterns.

Setting File Stripe Parameters

- File striping is controlled with the `lfs setstripe` command

```
lfs setstripe –c 3 –S 1M data
```

- File must not exist before running `lfs setstripe`

```
lfs setstripe: setstripe error for 'data': stripe already set
```

- Other striping options
  - `-i|--index` = Choose starting OST index for striping (default = -1)
  - `-o|--ost` = Specify OST indices to use
  - `-p|--pool` = Choose OSTs from the specified OST pool (default = none)

4 OAK RIDGE National Laboratory

File layouts are controlled using the "lfs setstripe" command. The command shown here sets a layout that matches the diagram on the previous slide.

Note: The file must not exist prior to running this command otherwise an error will result.

Many other striping options exist, but most of those are use for more advanced layouts (which we will discuss in a little while). Some other options for standard file layouts are –i, -o, and –p. Typically users should not use –i or –o unless they really know what they are doing. The –p option might be useful to normal users if the file systems has different tiers of storage.

For the –i option, a value of -1 indicates that Lustre should chose the starting OST.

Viewing File Stripe Parameters

- Use `lfs getstripe` to view the layout for a file

```
[tmp]# lfs getstripe data
data
lmm_stripe_count:   3
lmm_stripe_size:    1048576
lmm_pattern:        raid0
lmm_layout_gen:     0
lmm_stripe_offset: 2
        obdidx           objid           objid            group
             2           29564          0x737c       0xb00000417
             7           29644          0x73cc       0xc00000403
             6           29662          0x73de       0xe40000414
```

Allocated OSTs

5

OAK RIDGE National Laboratory

All file layouts can be viewed using "lfs getstripe". Several fields are printed, and we can see the stripe count and stripe size that were chosen for "lfs setstripe" (highlighted in yellow). The output also shows us exactly which OSTs were assigned to the file.

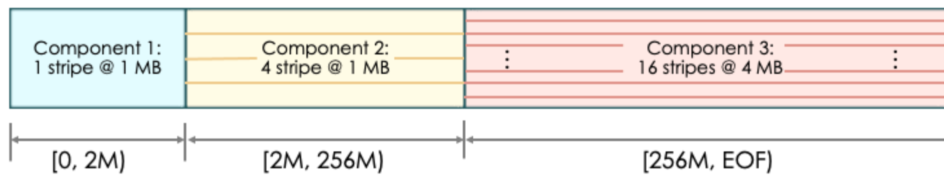
Notice that the stripe offset matches the index of the first OST.

Progressive File Layouts (PFL)

OAK RIDGE
National Laboratory

Progressive File Layouts (PFL)

- PFL allows more control over a file's layout
  - Normal layout applies single stripe count and stripe size to entire file
  - PFL can use different normal layouts on different parts of the file (making it one of the types of composite layouts)
  - Potentially optimize IO for files with non-uniform data structures



7 OAK RIDGE National Laboratory

You can think of PFL as an array of normal layouts that apply to different sections of the same file. This gives much greater control of a file's layout if needed.

This can also be used by sys admins to mitigate the possibility of a user filling up an ost. The admin can define a default PFL layout that increases the stripe size as the file gets bigger.

Creating PFL Layouts

- The `-E|--component-end` option is used to denote different components in the layout
- General format:

```
lfs setstripe -E end1 <stripe_opts> -E end2 <stripe_opts> … <file>
```

- Example:

```
lfs setstripe  -E 2M -c 1 -S 1M \
               -E 256M -c 4 -S 1M \
               -E -1 -c 16 -S 4M \      <- Can also use -E eof
               data.txt
```

8 OAK RIDGE National Laboratory

The ‘lfs setstripe’ command is used with the –E option to define PFL layouts. The –E option defines the end of the extant where the subsequent file stripe options apply.

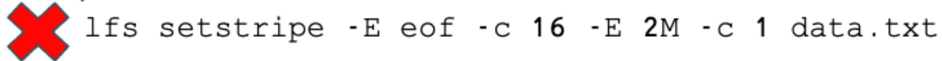
All the normal layout stripe options should be available to each component.

A “-1” or “eof” can be used to denote the end of the file.

Creating PFL Layouts (cont.)

- Components must be specified in order

  First component starts at offset = 0, next component starts where previous component ends



```
lfs setstripe -E eof -c 16 -E 2M -c 1 data.txt
```

- First component inherits default values from parent/root directory. Later components inherit values from previous component.

```
lfs setstripe -E 2M -c 1 -S 1M -E 64M -c 4 -E eof –S 4M
```

-E 2M -c 1 -S 1M	-E 64M -c 4	-E eof –S 4M
stripe count = 1 stripe size = 1M	stripe count = 4 stripe size = 1M	stripe count = 4 stripe size = 4M

OAK RIDGE National Laboratory

Lazy Initialization

- Lustre always allocates OSTs for the first component, but only allocates OSTs for other components when needed.

```
#> lfs getstripe data.txt
data.txt
  lcm_layout_gen:     3
  lcm_mirror_count:  1
  lcm_entry_count:   3
    lcme_id:               1
    lcme_mirror_id:        0
    lcme_flags:            init
    lcme_extent.e_start: 0
    lcme_extent.e_end:     2097152
      lmm_stripe_count:  1
      lmm_stripe_size:     1048576
      lmm_pattern:         raid0
      lmm_layout_gen:      0
      lmm_stripe_offset: 6
      lmm_objects:
      - 0: { l_ost_idx: 6, l_fid: [0xe40000414:0x7489:0x0] }
```

Viewing a PFL layout is done in the same way as a normal layout: using 'lfs getstripe'.

The layout shown here corresponds to the 3-component PFL layout we defined for the diagram 2 slides back. (This slide only shows the first of the three components)

The red-highlighted line show the field that contains the number of components in the file. The yellow-highlighted lines show general parameters for the first component (component ID, start/end points for the component, and whether the component is initialized or not.). The green-highlighted lines show the layout information for the first component (stripe count, stripe size, and allocated osts).

Lazy Initialization (cont.)

```
lcme_id:              2
lcme_mirror_id:       0
lcme_flags:           0
lcme_extent.e_start:  2097152
lcme_extent.e_end:    268435456
  lmm_stripe_count:   4
  lmm_stripe_size:    1048576
  lmm_pattern:        raid0
  lmm_layout_gen:     0
  lmm_stripe_offset:  -1

lcme_id:              3
lcme_mirror_id:       0
lcme_flags:           0
lcme_extent.e_start:  268435456
lcme_extent.e_end:    EOF
  lmm_stripe_count:   16
  lmm_stripe_size:    4194304
  lmm_pattern:        raid0
  lmm_layout_gen:     0
  lmm_stripe_offset:  -1
```

Components not initialized

No OSTs assigned

11 OAK RIDGE National Laboratory

This slide shows the other 2 components of the PFL layout.

Notice that neither of these components is initialized immediately after the file is created (as is evidenced by the lcme_flags=0 and lmm_stripe_offset=-1).

Yellow-highlighted fields show component-level parameters while the green-highlighted fields show the layout striping parameters for each component.

Lazy Initialization (cont.)

- Writing data will cause components to be initialized on-the-fly

```
# dd if=/dev/zero of=data.txt bs=1M count=4
```

- After 2M is written, Lustre initializes second component

```
lcme_id:             2
lcme_mirror_id:      0
lcme_flags:          init
lcme_extent.e_start: 2097152
lcme_extent.e_end:   268435456
  lmm_stripe_count:  4
  lmm_stripe_size:   1048576
  lmm_pattern:       raid0
  lmm_layout_gen:    0
  lmm_stripe_offset: 7
  lmm_objects:
  - 0: { l_ost_idx: 7, l_fid: [0xc00000403:0x746e:0x0] }
  - 1: { l_ost_idx: 8, l_fid: [0xc4000040e:0x7457:0x0] }
  - 2: { l_ost_idx: 12, l_fid: [0xe80000419:0x2a75:0x0] }
  - 3: { l_ost_idx: 1, l_fid: [0xac000040e:0x74c1:0x0] }
```

12 OAK RIDGE National Laboratory

When a client attempts to write data to a component that has not yet been initialized, I/O will pause until Lustre allocates OSTs for the next component.

This snippet from "lfs getstripe" shows that the second component was initialized after writing 4MB to the file. The third component (not shown) remains uninitialized. The yellow-highlighted fields show some key parameters:

- lcme_flags is now set to "init" instead of "0"
- 4 osts have been allocated to the component
- lmm_stripe_offset matches the index of the first allocated ost

Once allocated, the OSTs for a component remain allocated. So truncating a file will not cause later components to unallocated their OSTs.

Dynamic Layout Changes

- With lazy initialization, only the first component is required to be specified when the file layout is set
- Other components can be added (and even deleted) dynamically using `lfs setstripe`
- There are some caveats:
  - Components can only be deleted starting from the last one
  - Deleting a component will cause all data in that component to be lost
  - Cannot write past the end of the last component

OAK RIDGE National Laboratory

PFL does not allow "holes" in the layout, so the only component that can be deleted is the last component.

Be careful when deleting a component. If there is data in that component, you will not be prompted before it is deleted.

Dynamic Layout Example

```
[tmp]# lfs setstripe -E 2M -c 1 data.txt

[tmp]# dd if=/dev/zero of=data.txt bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00139419 s, 752 MB/s

[tmp]# lfs setstripe --component-add -E 10M -c 4 data.txt

[tmp]# lfs setstripe --component-add -E -1 -c 8 data.txt

[tmp]# dd if=/dev/zero of=data.txt bs=1M count=20
20+0 records in
20+0 records out
20971520 bytes (21 MB, 20 MiB) copied, 0.0671822 s, 312 MB/s

[tmp]# ls -lh data.txt
-rw-r--r-- 1 root root 20M May  2 01:40 data.txt
```

Create initial component…

…then add two more

Data extends Into the third component

14 OAK RIDGE National Laboratory

We can start the file with just a single component and then immediately begin adding data to it. But before we reach the end of the first component, we extend the layout with two more components that cover the rest of the file. After that, we can write data as far as we want into the file.

Dynamic Layout Example (cont.)

```
[tmp]# lfs setstripe --component-del -I 3 data.txt
```

component index* — Deleting last component…

```
[tmp]# ls -lh data.txt
-rw-r--r-- 1 root root 10M May  2 01:40 data.txt
```

…truncates the file and destroys data in the deleted component

```
[tmp]# dd if=/dev/zero of=data.txt bs=1M count=20
dd: error writing 'data.txt': No data available
11+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.00900609 s, 1.2 GB/s
```

Can't write past end of the last component

*Not necessarily sequential. Check `lcme_id` field in output from `lfs getstripe`

15 OAK RIDGE National Laboratory

The lcme_id field of each component appears to be tied to the lcm_layout_gen field which contains a generation number that increments whenever there is a change to the layout. This prevents collisions or re-use of the same component index as the layout changes.

Other Useful Commands

- Component-related options for `lfs getstripe`

  (Check man page for full set of options)

```
lfs getstripe --component-count <file>
lfs getstripe –I2 <file>
lfs getstripe --component-flag=init <file>
```

List number of components

List component with id=2

List only initialized components

- Use the lfs migrate command to change a normal layout to PFL (and vice versa)

```
lfs setstripe -c 1 data.txt
dd if=/dev/zero of=data.txt bs=1M count=100
lfs migrate -E 2M -c 1 -E 20M -c 4 -E -1 -c 16 data.txt
```

16 OAK RIDGE National Laboratory

These are just a few of the component-specific options available for lfs getstripe. (NOTE: There seems to be a mistake in the online manual. It shows using the "-I" option to list the indices of all components in a file. In my testing, it only listed the last initialized component.)

NOTE: Make sure there is not space between "-I" and "2". Otherwise you will get an error.

Important Note About File Appends

- Components are normally only initialized when data is written to them…except in the case of file appends

```
[tmp]# lfs setstripe -E 100M -c 1 -E 10G -c 4 -E -1 -c -1 data.txt
[tmp]# lfs getstripe data.txt
[tmp]# dd if=/dev/zero of=file.txt bs=1M count=1
[tmp]# lfs getstripe data.txt
[tmp]# echo "This is a test" >> data.txt
[tmp]# lfs getstripe data.txt
```

Only first component is initialized

All components Initialized!!

17 OAK RIDGE National Laboratory

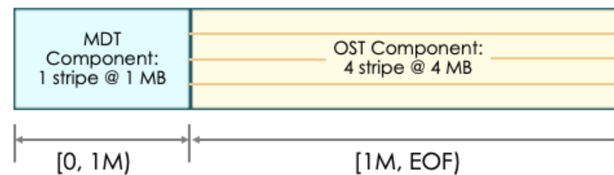
Appending to a file will cause all components to be initialized even if there is no data in those components.

Data on MDT (DoM)

OAK RIDGE
National Laboratory

DoM Basics

- DoM layouts are intended to improve small file I/O performance by placing all (or part) of the file on the MDT
- A DoM layout is a composite layout (and is actually just a special case of PFL)
  - Only the first component resides on the MDT
  - The first component always has stripe_count = 1
  - The MDT used for the first component is the same MDT that stores the inode for the file



19 OAK RIDGE National Laboratory

There could be uses for DoM even for larger files (ex - a structured file that begins with a small header describing the layout for the rest of the file's data).

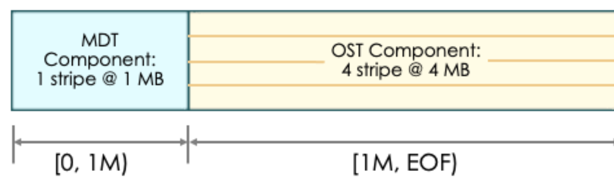
Creating DoM Layout

- DoM layout are created in a similar fashion to PFL layouts

```
lfs setstripe -E <end1> -L mdt -E <end2> [stripe_opts]…
```

- Example:

```
lfs setstripe -E 1M -L mdt -E eof -c 4 -S 4M dom_file
```



20 OAK RIDGE National Laboratory

The '--layout' option could also be used instead of '-L'

Displaying DoM Layout

```
[tmp]# lfs getstripe dom_file

dom_file
  lcm_layout_gen:     2
  lcm_mirror_count:   1
  lcm_entry_count:    2
    lcme_id:              1
    lcme_mirror_id:       0
    lcme_flags:           init
    lcme_extent.e_start:  0
    lcme_extent.e_end:    1048576
      lmm_stripe_count:   0
      lmm_stripe_size:    1048576
      lmm_pattern:        mdt
      lmm_layout_gen:     0
      lmm_stripe_offset:  0

    lcme_id:              2
    lcme_mirror_id:       0
    lcme_flags:           0
    lcme_extent.e_start:  1048576
    lcme_extent.e_end:    EOF
      lmm_stripe_count:   4
      lmm_stripe_size:    4194304
      lmm_pattern:        raid0
      lmm_layout_gen:     0
      lmm_stripe_offset:  -1
```

DoM layout looks nearly identical to a PFL layout

stripe size = extent end

Pattern is 'mdt' instead of 'raid0'

21

Since a DoM layout is just a type of PFL layout, they are displayed almost identically. The key differences are:

1) Stripe size will always match the extent end (green-highlighted fields)
2) The stripe count will be '0', although more accurately it is '1' (red-highlighted field)
3) The pattern is 'mdt' instead of 'raid0' (blue-highlighted field)

This doesn't seem to show which mdt the first component resides on. You can get the mdt for the file using "lfs getstripe –m <file>", and this same mdt will contain the first component of the file.

DoM for Sys Admins

- System administrators can control the maximum size of the DoM components
- On the MDS server, run

```
lctl set_param –n lod.*MDT<index>*.dom_stripesize=<max>
```

Temporary

or

```
lctl set_param -P lod.*MDT<index>.lod.dom_stripesize=<max>
```

Persistent

- For the `dom_stripesize` value:
  - Default value is 1 MB
  - No smaller than 64 KB and no larger than 1 GB
  - Must be 64 KB aligned

22 OAK RIDGE National Laboratory

The dom_stripesize values are set on a per-MDT basis. So it is possible for two MDTs in the same file system to have different values.

If dom_stripesize is set to a value below 64K, the lctl command will not return an error and it will appear to have worked. But when you query the value of the parameter, you'll find that dom_stripesize is just set to its minimum value of 64 KB.

However, if you try to set dom_stripesize to a value above 1 GB, you wil receive a "numerical result out of range" error, and its value will remain unchanged.

If the value is not 64 KB aligned, it will silently round down the value to the nearest 64 KB aligned value and use that as the value.

Dom for Sys Admins (cont.)

- DoM can be disabled by setting `dom_stripesize` to 0
  - This can be done on a per-MDT basis as well
- This will disable DoM component creation for any new files or layouts
  - Existing files with DoM components will remain unchanged
  - If a directory has a default layout defined that contains a DoM component, new files in that directory can still be created with DoM components
- DoM files can be identified using `lfs find`

```
lfs find <dir> -L mdt
```


OAK RIDGE National Laboratory

File Creation with DoM Disabled

If DoM is disabled, attempting to create a DoM file will not fail. The layout just gets automatically altered.

```
[tmp]# lfs setstripe -E 1M -L mdt -E eof -c 2 dom_file

[tmp]# lfs getstripe dom_file
dom_file
  lcm_layout_gen:    1
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0xb8000041e:0x743c:0x0] }
      - 1: { l_ost_idx: 10, l_fid: [0xcc000041a:0x73be:0x0] }
```


OAK RIDGE National Laboratory

File Level Redundancy (FLR)

25 OAK RIDGE National Laboratory

File Level Redundancy (FLR)

- FLR allows users to define one or more mirrors for a file to provide extra data protection.
- When writing to a mirrored file, only one of the mirrors is updated. Other mirrors are marked as stale and need to be resynced.
  - This is the “FLR Delayed Write” implementation
- All mirrors (that are not stale) can be used for reading data
  - Can be used to improve read performance for files that are accessed by many processes in parallel

OAK RIDGE National Laboratory

Creating FLR Layouts

- Unlike other file layouts, this one does not use `lfs setstripe`. Instead, there is a special `lfs mirror` command.

```
lfs mirror create -N[count] [stripe_opts] [--flags=<flags>] … <file>
```

- The mirrors can be either normal layouts, composite layouts, or a mixture of both

```
lfs mirror create -N2 --flags=prefer -c 2 -N -E 10M -c 1 -E eof -c 4 data.txt
```

Two mirrors have the same normal layout

Third mirror uses PFL layout

OAK RIDGE National Laboratory

The "-N" option without a value is equivalent to "-N1". The long option --mirror-count can also be used instead of –N.

The stripe options are the same as those used by normal and PFL layouts. The ones most commonly used will probably be stripe count, stripe size, and ost pool.

The only option supported by --flags is "prefer". This gives Lustre a hint about which mirrors it should try to use for I/O, but there is no guarantee that the preferred mirrors will be used.

Displaying FLR Layouts

```
[tmp]# lfs getstripe data.txt
  lcm_mirror_count:  3
  lcm_entry_count:   4
    lcme_mirror_id:          1
    lcme_flags:              init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:       EOF
      lmm_stripe_count:  2
      lmm_stripe_size:       1048576
      lmm_objects:
      - 0: { l_ost_idx: 7, l_fid: [0xc00000403:0x7479:0x0] }
      - 1: { l_ost_idx: 6, l_fid: [0xe40000414:0x7493:0x0] }

    lcme_mirror_id:          2
    lcme_flags:              init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:       EOF
      lmm_stripe_count:  2
      lmm_stripe_size:       1048576
      lmm_objects:
      - 0: { l_ost_idx: 15, l_fid: [0xdc0000419:0x2acf:0x0] }
      - 1: { l_ost_idx: 9, l_fid: [0xc80000405:0x73e4:0x0] }
```

Number of mirrors

Note: Some fields omitted for brevity

Each mirror has unique ID

28 OAK RIDGE National Laboratory

Even though FLR uses a different command to create the layout, it is still viewed using "lfs getstripe". (Some fields are omitted for clarity. One field in particular that is missing is the lcme_id field that uniquely identifies each component.)

The output show there are 3 mirrors, but there are a total of 4 components since the third mirror has a PFL layout. This slide also show details on the first two mirrors. They use the same layout (stripe count = 2 across the entire file) but they have different OSTs assigned to them.

Note that they are identified by different lcme_mirror_id fields.

Yellow-highlighted fields describe the number of components/mirrors in the file. The blue- and red-highlighted fields are for the first and second component respectively. Note that the values are identical except for the lcme_mirror_id field which uniquely identifies each mirror.

Displaying FLR Layouts (cont.)

```
lcme_mirror_id:        3
lcme_flags:            init
lcme_extent.e_start:   0
lcme_extent.e_end:     10485760
  lmm_stripe_count:    1
  lmm_stripe_size:     1048576
  lmm_objects:
  - 0: { l_ost_idx: 0, l_fid: [0xe00000416:0x73f9:0x0] }

lcme_mirror_id:        3
lcme_flags:            0
lcme_extent.e_start:   10485760
lcme_extent.e_end:     EOF
  lmm_stripe_count:    4
  lmm_stripe_size:     1048576
  lmm_stripe_offset:   -1
```

Only first component of PFL is initialized

All components in the same mirror have identical `lcme_mirror_id` fields

29

OAK RIDGE National Laboratory

The last two components have the same lcme_mirror_id because they are part of the PFL layout that comprises the third mirror.

The green-highlighted fields are intended to emphasize two things:

1) These two component are part of the same mirror
2) The extents are non-overlapping and cover the entire range of the file

Resync and Verify

- When writing data, the `lcme_flags` field for some components may change to indicate they are out of sync:

```
lcme_flags: init,prefer
```

data written

```
lcme_flags: init,stale,prefer
```

- This is fixed by running

```
lfs mirror resync <file>
```

- Mirrored data can also be checked for consistency

```
lfs mirror verify <file>
```


OAK RIDGE National Laboratory

Extending a Mirrored File

- Additional mirrors can be added to an existing file

```
lfs mirror extend -N[mirror_count] [stripe_options] … <file>
```

  - If the file is not a mirrored file already, it will be converted to one
  - Existing data is copied to the new mirror
- An existing file can also be added as a mirror to another file

```
lfs mirror extend [--no-verify] -N -f victim_file <file>
```

  - The user needs to ensure that victim_file contains the same data

31 OAK RIDGE National Laboratory

When using a victim file, Lustre will verify that the data matches or else it returns an error. If the user is sure that the files match, the --no-verify option can be added to skip this verification check.

If successful, the victim file will be removed from the file system namespace.

Splitting a Mirrored File

- A specified mirror can be split from a mirrored file into its own separate file

```
lfs mirror split -mirror-id <ID> [-f <new_file>] <mirrored_file>
```

- If the `-f` option is not used, then the default name for the new file will be `<mirrored_file>.<mirror_id>`
- To destroy the mirror instead of splitting into its own file, just use the `-d|--destroy` option instead of `-f`

OAK RIDGE National Laboratory

Self-Extending Layouts (SEL)

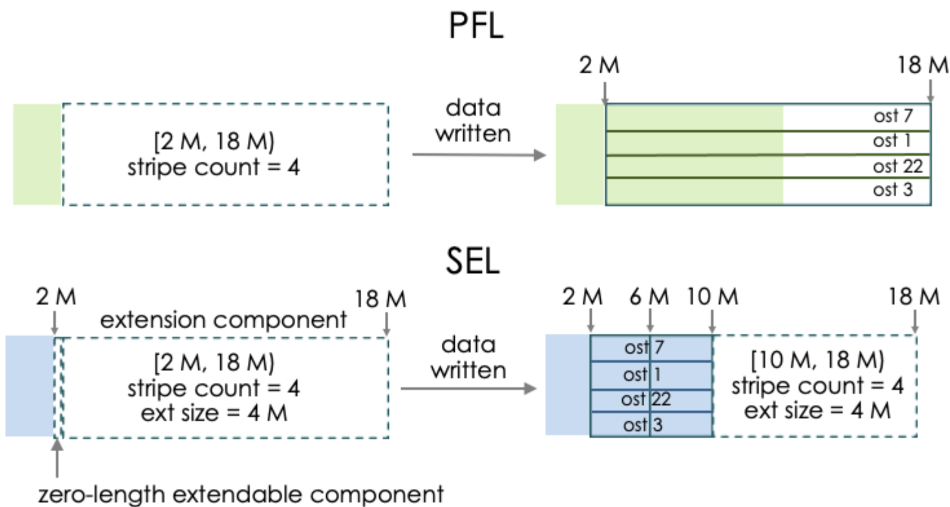
Self-Extending Layouts

- SEL is an extension of the PFL feature that allows the MDS to change the PFL layout dynamically if it detects OSTs running low on space
- PFL delays instantiation of some components
- SEL splits non-instantiated components into two parts
  1. An extendable component that is a regular PFL component covering a part of the region
  2. An extension component that is never instantiated and covers the remainder of the region

OAK RIDGE National Laboratory

PFL delays instantiation, but when it finally does instantiate the component, that layout is used for the entire region without changes.

SEL basically turns a single component into two sub-components, although the second sub-component is never instantiated and just acts as a placeholder until the MDS decides what action to take when expanding the first sub-component.



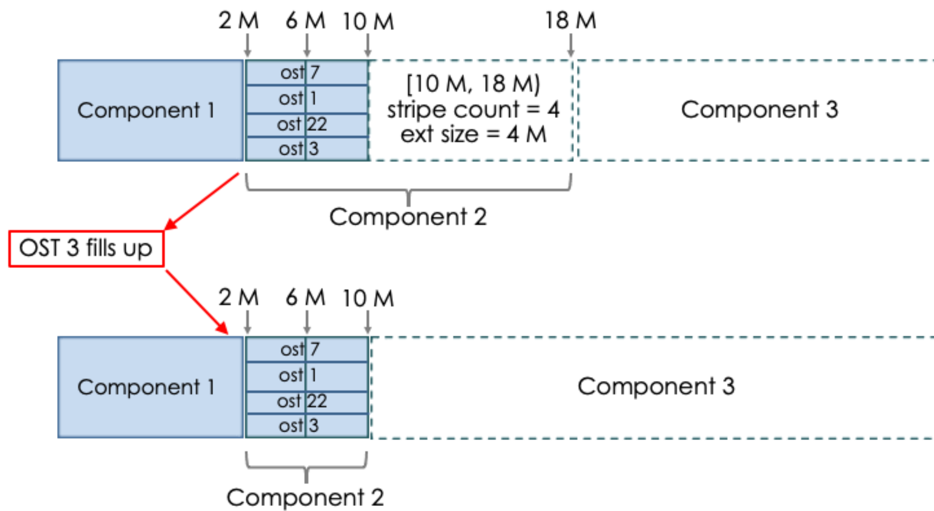
SEL alters the components on-the-fly. If the OSTs do not run low on space, then eventually the extension component shrinks to zero. At that point, the component is no different than the PFL layout.

SEL would probably be used by sys admins more than normal users. A sys admin could use SEL to define a layout that starts using storage on a "fast tier" that has smaller capacity. If the fast tier fills up, the file automatically switches to using at larger capacity "slow tier".

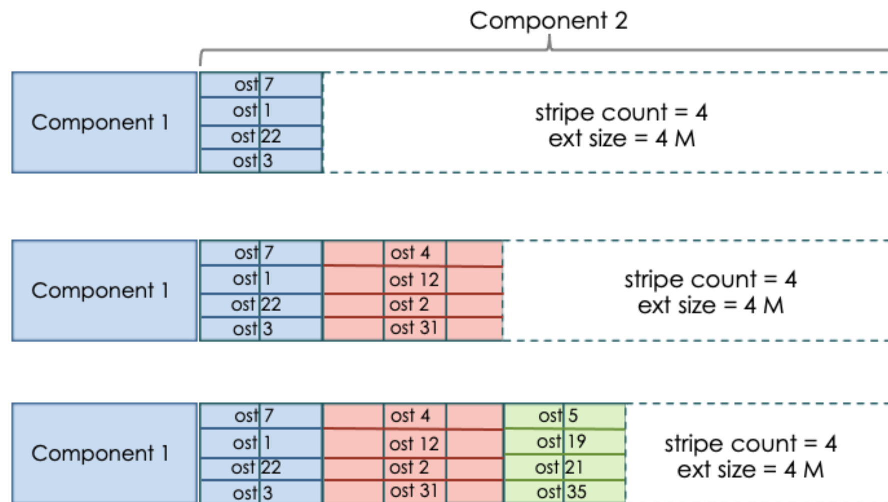
Extension Policies

- The benefit of SEL comes from what it does when an OST starts to fill up. There are four policies that handle various cases:
  1. Extension – When OSTs in current component are not low on space, continue using them (illustrated in the previous slide)
  2. Spill Over – If current component is not the last component, and one of the current OSTs is low on space, switch to next SEL component
  3. Repeating – If current component is the last component and one of the OSTs is low on space, create a new component with the same layout as the current component (but using different OSTs)
  4. Forced extension – If current component is the last component, and an attempt to repeat the layout fails due to low space, just keep using the current OSTs

OAK RIDGE National Laboratory



If an OST runs low on space, just abort the current component early and start using the next component. When component 3 gets instantiated, the MDS should choose osts that are not low on space.



If the SEL component is the last component, you can't just abandon it early and move to the next component. Instead, the MDS will extend the component using the same layout, but with a different sent of OSTs. It can keep doing this is OSTs keep running out of space.

Creating a Self-Extending Layout

- Use the same command as for PFL…

```
lfs setstripe -E end1 <stripe_opts> -E end2 <stripe_opts> … <file>
```

- …but add a `-z |--extension-size` option with the other stripe options

```
lfs setstripe -E 1G -z 64M -c 1 -E -1 -z 256M -c 4 data.txt
```

39 OAK RIDGE National Laboratory

From my testing, it looks like the minimum value for the extension size is 64MB, but it's not clear why. This doesn't appear to be documented in the Lustre manual.

Viewing Self-Extending Layout

Note: Only some of the output fields are shown for brevity

```
[tmp]# lfs getstripe data.txt
data.txt
  lcm_entry_count:   4
    lcme_id:              1
    lcme_flags:           init
    lcme_extent.e_start: 0
    lcme_extent.e_end:    67108864
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_stripe_offset: 10
      lmm_objects:
      - 0: { l_ost_idx: 10, l_fid: [0xcc000041a:0x73c2:0x0] }

    lcme_id:              2
    lcme_flags:           extension
    lcme_extent.e_start: 67108864
    lcme_extent.e_end:    1073741824
      lmm_stripe_count:  0
      lmm_extension_size: 67108864
      lmm_stripe_offset: -1
```

Denotes extension component

40 OAK RIDGE National Laboratory

The green-highlighted fields illustrate that the initial range of the extendable component matches the extension size of the extension component.

The red-highlighted fields illustrate three things:

1) The lcme_flags field is set to "extension" for the non-instantiated extension component
2) The extension component always starts from the same spot that the extendable component ends
3) The end of the extension component matches the end of the region defined in the original "lfs setstripe" command

Viewing Self-Extending Layout (cont.)

```
lcme_id:             3
lcme_flags:          0
lcme_extent.e_start: 1073741824
lcme_extent.e_end:   1073741824
  lmm_stripe_count:  4
  lmm_stripe_size:   1048576
  lmm_stripe_offset: -1

lcme_id:             4
lcme_flags:          extension
lcme_extent.e_start: 1073741824
lcme_extent.e_end:   EOF
  lmm_stripe_count:  0
  lmm_extension_size: 268435456
  lmm_stripe_offset: -1
```

Zero length extendable component

As extendable component grows, the extension component shrinks

41 OAK RIDGE National Laboratory

The green-highlighted fields show that, prior to data being written, the extendable component starts out as a zero-length component. The red-highlighted fields denote some of the important fields in the non-instantiated extension component.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

OAK RIDGE
National Laboratory

Questions ?

43 OAK RIDGE National Laboratory