Self-Extending Layout (SEL) Operations Manual, v0.2

Author: Vitaly Fertman

The Lustre Self-Extending Layout (SEL) feature is an extension of the Progressive File Layout which allows MDS to change the defined PFL layout dynamically. MDS is monitoring the used space on OSTs and SEL lets it to swap the current file OSTs when they are low on space, this allows to avoid ENOSPC problems for SEL files when applications are writing to them.

Whereas the PFL delays the instantiation of some components until an IO operation occurs on this region, SEL allows to split such non-instantiated components on two parts, in fact individual PFL components: an “extendable” component and an “extension” component. The extendable component is a regular PFL component, covering just a part of the region, which is small originally. The extension (or SEL) component is a new component type which is always non-instantiated and unassigned, covering the other part of the region. When a write reaches this unassigned space (and the client calls MDS to have it instantiated) MDS makes a decision if to grant some space to the extendable component (the granted region moves from the head of the extension component to the tail of the extendable component, thus the extendable component grows and the SEL one is shortened) and therefore allows the file to continue on the same OSTs; or (in case of low on space of at least one of the current OSTs) to modify the layout to switch to a new component on new OSTs. In particular, it lets IO automatically to spill over to a large HDD OST pool once a small SSD OST pool is getting low on space.

The default extension policy modifies the layout in the following ways:

1. Extension: continue on the same OSTs – used when low on space condition does not occur on any of the OSTs of the current component; a particular extent is granted to the extendable component;
2. Spill over: switch to next component OSTs – it is used only for not the last component when at least 1 of the current OSTs is low on space; the whole region of the SEL component moves to the next component and the SEL component is removed in its turn.
3. Repeating: a new component with the same layout but on free OSTs is created – it is used only for the last component when at least 1 of the current OSTs is low on space; a new component has the same layout but instantiated on different OSTs (from the same pool) which have enough space;
4. Forced extension: continue with the current component OSTs despite the low on space condition – it is used only for the last component when a repeating attempt detected low on space condition as well - spillover is impossible and there is no sense in the repeating.

Note: the SEL feature does not require clients to understand the SEL format of already created files, only the MDS support is needed which is introduced in Lustre 2.13. However, old clients will have some limitations as the Lustre tools will not support it.

New commands are introduced to the Lustre tools to operate SEL files, below are the new commands and examples for the lfs setstripe, lfs find, lfs getstripe

1. lfs setstripe

lfs setstripe command is extended for SEL feature.

1.1. Create a SEL file

lfs setstripe -E end1 [STRIPE_OPTIONS] ... <directory|file>

STRIPE OPTIONS:
--extension-size, --ext-size, -z <ext_size>

The -z option is added to specify the size of the region which is granted to the extendable component on each iteration. While declaring any component, this option turns the declared component to a pair of components: extendable and extension ones.

Example 1. Create a SEL file.

```
# lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/file
```

This command creates 2 pairs of extendable and extension components:
[0, 64M], SEL [64M, 1G], [1G, 1G], SEL [1G, EOF].

Note: As usually, only the 1st PFL component is instantiated at the creation time, thus it is immediately extended to the extension size (64M for the 1st component), whereas the 3rd component is left 0-length.

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
  lcm_layout_gen:    4
  lcm_mirror_count:  1
  lcm_entry_count:   4
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   67108864
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

    lcme_id:             2
    lcme_mirror_id:      0
    lcme_flags:          extension
    lcme_extent.e_start: 67108864
    lcme_extent.e_end:   1073741824
      lmm_stripe_count:  0
      lmm_extension_size: 67108864
      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: 1073741824
  lcme_extent.e_end:   1073741824
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1

  lcme_id:            4
  lcme_mirror_id:     0
  lcme_flags:         extension
  lcme_extent.e_start: 1073741824
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  0
    lmm_extension_size: 268435456
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

Note: As you can see the SEL components are marked by the “extension” flag and lmm_extension_size field keeps the specified extension size.

Example 2. Set default SEL layout to an existing directory

Similar to PFL, it is possible to define a default SEL layout for a directory. After that, all the files created will inherit this layout by default. This does not differ from the PFL operations.

Example3. Extension.

Suppose a SEL file is created:
```
# lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/file
```

i.e. with the following components: [0, 64M], SEL [64M, 1G], [1G, 1G], SEL [1G, EOF].

Suppose there is a write which crosses the end of the 1st component (64M), the result layout will be: [0, 128M], SEL [128M, 1G], [1G, 1G], SEL [1G, EOF]
Suppose another IO proceeds and crosses the end of the 1st component (64M) again:
[0, 192M], SEL [192M, 1G], [1G, 1G], SEL [1G, EOF]

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
  lcm_layout_gen:    6
  lcm_mirror_count:  1
  lcm_entry_count:   4
    lcme_id:            1
    lcme_mirror_id:     0
    lcme_flags:         init
    lcme_extent.e_start: 0
```

```
  lcme_extent.e_end:   201326592
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

  lcme_id:           2
  lcme_mirror_id:    0
  lcme_flags:        extension
  lcme_extent.e_start: 201326592
  lcme_extent.e_end:   1073741824
    lmm_stripe_count:  0
    lmm_extension_size: 67108864
    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: 1073741824
  lcme_extent.e_end:   1073741824
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1

  lcme_id:           4
  lcme_mirror_id:    0
  lcme_flags:        extension
  lcme_extent.e_start: 1073741824
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  0
    lmm_extension_size: 268435456
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

Example4. Spillover.

In case OST0 is low on space, a spillover happens, when the full region of the SEL component is added to the next component, e.g. in the example above after 2 successful extensions and spillover it will look like:

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
  lcm_layout_gen:    7
  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:   201326592
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

    lcme_id:             3
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 201326592
    lcme_extent.e_end:   469762048
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
      lmm_objects:
      - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }

    lcme_id:             4
    lcme_mirror_id:      0
    lcme_flags:          extension
    lcme_extent.e_start: 469762048
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  0
      lmm_extension_size: 268435456
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
```

Note: Despite the fact the 3rd component was [1G, 1G] originally, while it is not instantiated, instead of getting extended backward, it is moved backward to the start of the previous SEL component (192M) and extended on its extension size (256M) from that position, thus it becomes [192M, 448M].

Example5. Repeating.

Suppose in the example above, OST0 got enough free space back but OST1 is low on space, the following write to the last SEL component leads to a new component allocation before the SEL component, which repeats the previous component layout but instantiated on free OSTs:

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
  lcm_layout_gen:    9
  lcm_mirror_count:  1
  lcm_entry_count:   4
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   201326592
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

    lcme_id:             3
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 201326592
    lcme_extent.e_end:   469762048
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
      lmm_objects:
      - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }

    lcme_id:             8
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 469762048
    lcme_extent.e_end:   738197504
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    65535
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x6:0x0] }
```

```
  lcme_id:           4
  lcme_mirror_id:    0
  lcme_flags:        extension
  lcme_extent.e_start: 738197504
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  0
    lmm_extension_size: 268435456
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

Example 7. Forced extension.

Suppose in the example above, both OST0 and OST1 are low on space, the following write to the last SEL component will behave as an extension as there is no sense to repeat.

```
# ./lustre/utils/lfs getstripe /mnt/lustre/file
/mnt/lustre/file
  lcm_layout_gen:    11
  lcm_mirror_count:  1
  lcm_entry_count:   4
    lcme_id:           1
    lcme_mirror_id:    0
    lcme_flags:        init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   201326592
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

    lcme_id:           3
    lcme_mirror_id:    0
    lcme_flags:        init
    lcme_extent.e_start: 201326592
    lcme_extent.e_end:   469762048
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
      lmm_objects:
      - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }

    lcme_id:           8
```

```
  lcme_mirror_id:     0
  lcme_flags:         init
  lcme_extent.e_start: 469762048
  lcme_extent.e_end:   1006632960
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    65535
    lmm_stripe_offset: 0
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x6:0x0] }

  lcme_id:            4
  lcme_mirror_id:     0
  lcme_flags:         extension
  lcme_extent.e_start: 1006632960
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  0
    lmm_extension_size: 268435456
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

2. lfs getstripe

lfs getstripe is extended to operate SEL files. When getting the striping of the file, extension components are marked by the “extension” flag and lmm_extension_size field has the extension size specified at the time of the “lfs setstripe”.

--extension-size|--ext-size|-z

The -z option is added to print the extension size in bytes. For composite files this is the extension size of the first extension component. If a particular component is identified by other options (--component-id, --component-start, etc), this component extension size is printed.

Example.

```
# ./lustre/utils/lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/file
# ./lustre/utils/lfs getstripe -z -I2 /mnt/lustre/file
67108864
```

3. lfs find

lfs find is extended to operate on SEL files. It can be used to search for the files that match the given SEL extension size.

--extension-size|--ext-size|-z [+-]ext-size[KMG]
[[!] --component-flags=extension]

The -z option is added to specify the extension size to search for. The files which have any component with the extension size matched the given criteria are printed out. As always “+” and “-“ signs are allowed to specify the least and the most size.

A new component flag is added “extension”. Only those files are printed which have at least one SEL component.

Note: The negative search for flags searches the files which **have** a non SEL component (not files which **do not have** a SEL component).

Example.

```
# lfs setstripe --extension-size 64M -c 1 -E -1 /mnt/lustre/file
# lfs find --comp-flags extension /mnt/lustre/*
/mnt/lustre/file
# lfs find ! --comp-flags extension /mnt/lustre/*
/mnt/lustre/file
# lfs find -z 64M  /mnt/lustre/*
/mnt/lustre/file
# lfs find -z +64M  /mnt/lustre/*
# lfs find -z -64M  /mnt/lustre/*
# lfs find -z +63M  /mnt/lustre/*
/mnt/lustre/file
# lfs find -z -65M  /mnt/lustre/*
/mnt/lustre/file
# lfs find -z 65M  /mnt/lustre/*
# lfs find ! -z 64M  /mnt/lustre/*
# lfs find ! -z +64M  /mnt/lustre/*
/mnt/lustre/file
# lfs find ! -z -64M  /mnt/lustre/*
/mnt/lustre/file
# lfs find ! -z +63M  /mnt/lustre/*
# lfs find ! -z -65M  /mnt/lustre/*
# lfs find ! -z 65M  /mnt/lustre/*
/mnt/lustre/file
```