Whamcloud

FLR Erasure Coding

Bobijam Xu, Whamcloud

Mirror vs. Erasure Coding

Whamcloud

- Space efficiency
  - Mirror redundancy requires at least 100% space overhead
  - EC provides more space-efficient data redundancy (typically 20-25%)
- CPU overhead
  - Mirror does not require more CPU to read or write
  - EC needs significant CPU to generate parity and to restore missing data

Design Overview

Whamcloud

- User-space interfaces for command line tools and library APIs
  - Handled similarly to mirror components
    - Generate EC in userspace based on data components
    - Write EC into separate stripe component
- Changes to CLIO infrastructure
  - Mark EC component(s) stale if data is modified
  - Handle data reconstruction in case of OST failure
    - Generate read IO for all available data stripes and EC stripes
    - Fill missing pages belonging to failed OSTs
- Changes to MDS
  - Understand EC layout component

lfs setstripe

- `lfs setstripe {--component-end|-E end1} [comp1_options]`
  `{--erasure-code} [-k number_of_data_devices ]`
  `{-m number_parity_devices} [-i start_ost_idx | -o ost_idx]`
  `[ -p pool_name]` `{--component-end|-E end2 } [comp2_options]`
  `[{--erasure-code} ...]` `{filename|directory}`

  - **`-k`** used to specify the number of data stripes used to compute the erasure code out of them
    - Default is to use all stripes for computing erasure code
  - **`-m`** specifies the number of code stripes to be use
    - Encoding takes *k* data devices calculates *m* code devices

Data Structure

Whamcloud

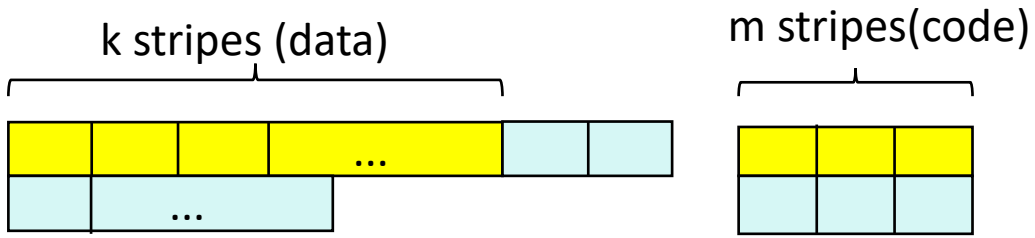
- On-disk component entry blob for code

```
struct lov_mds_md_v4 {
    __u32 lmm_magic;            // LOV_MAGIC_EC
    __u32 lmm_pattern;
    struct ost_id lmm_oi;
    __u32 lmm_stripe_size;
    __u16 lmm_stripe_count;     // total stripe device count, include data & code stripes
    __u16 lmm_layout_gen;
     char lmm_pool_name[LOV_MAXPOOLNAME + 1];
+   __u16 lmm_dstripe_count;    // data stripe count used in EC, k value
                                // must be <= lmm_stripe_count - lmm_cstripe_count
+   __u16 lmm_cstripe_count;    // code stripe count, m value
    struct lov_ost_data_v1 lmm_objects[0];
};
```

Different files for data and parity code

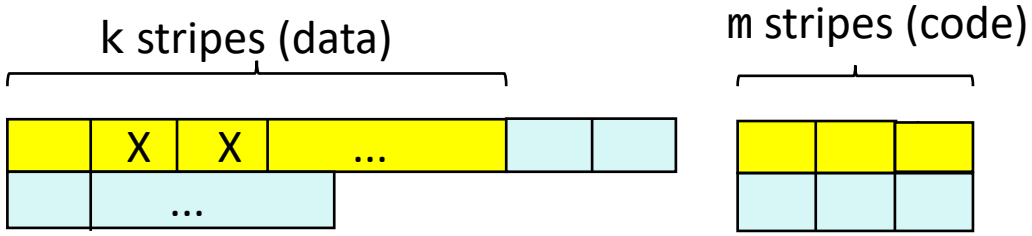
- Data page cache is reference by file and its index offset
- So parity code uses another mapping in memory to index pages
  - Use regular LOV IO offset calculation/code
  - Use CPU-optimized EC generation code

```
pages_per_stripe = stripe_size / PAGE_SIZE
code_page_index[i] = data_page_index / (k * pages_per_stripe) * m +
                     i * pages_per_stripe  (where i = 0 ... m-1)
```



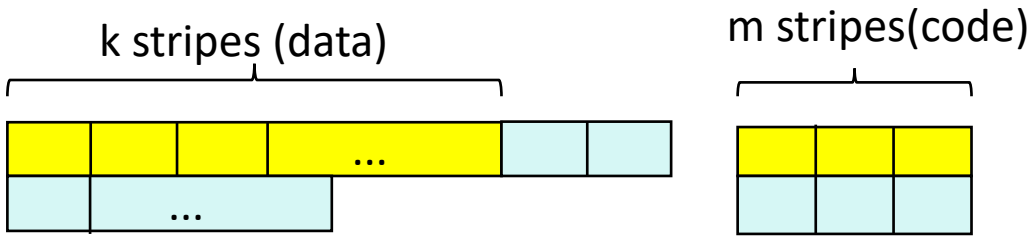
Read

- Encounters failed OST, check whether code component stale
- If EC component not STALE, read relevant data stripes and parity code stripes
  - In read I/O, how to build another I/O to read other file
  - Cannot read data if missing OSTs *and* EC component is STALE
- Calculate missing stripes and fill the corresponding page cache



EC Generation/Resync

- Create volatile parity code file
- Calculate and write parity code
- Mark code component uptodate
- Merge EC component to file



Whamcloud