
whamcloud

Lustre file striping across large number of OSTs

- Oleg Drokin

  Senior Engineer

  Whamcloud, Inc.

Current situation with striping

- Tens of thousands of OSTs possible
  - Existing systems with up to 1,300 OSTs
  - OST count is bound to grow
- 160 stripes per file max
  - Limit is due to the way striping information is stored
  - ext3 & ext4 have limited amount of "EA" space
  - 32(48) byte header + N stripes * 24 bytes object description
  - 160*24 + 32 = 3872 + filesystem overhead

Why do we need more stripes per file?

- Single file is limited in bandwidth to what 160 OSTs can provide
  - A big deal for apps that prefer to deal with a single file
- A single file size is limited to ~320T
  - Single object is limited to 2T-4k
  - Higher ext4 file size limit cannot be used for compatibility reasons.

Evolutionary growing the stripe count limit

- Evolutionary means:
  - No major code changes in Lustre
  - No protocol changes required to allow old clients to work with the filesystem
- Changes involved:
  - Ext3/4 to allow bigger EA space
  - Lustre buffer allocations change to accommodate bigger data size
  - Compatibility code to not send too big data to old clients
  - Server-side object destroys on unlink

Ext3/4 changes

- To allow larger EA data:
  - Allocate new inode
  - Store large EA data as file body of that inode
    - Allows for really large EA sizes if we want to
  - Original file inode points to this new EA inode
    - Specially encoded xattr block pointer
- Resulting filesystem is not backwards compatible with older ext3/4 code
- Agreement from ext4 maintainers to adopt this code if another feature is implemented.

Lustre buffer allocations

- No way to know required buffer size before contacting server
  - Must use pessimistic estimate of maximum number of OSTs
- Change large allocations to vmalloc
  - Linux does not like normal allocations of more than 2 pages
- All network buffer allocations must grow
  - We don’t know striping beforehand so must assume the worst
  - To contain buffer growth maximum stripe count limited to 1352
  - Max MDS request size changed from 5k to 32k
  - Max MDS reply size changed from 9k to 75k

Compatibility with older clients

- Old clients still have smaller buffers
  - Only report total RPC reply size
- A test during create to only stripe up to 160-way for small RPCs
- Other RPCs test if the reply would fit
  - Return –EFBIG if not
  - Reserved space for other buffers (like XATTRs) allows access to files striped more than 160-way

Unlink implications

- Unlink requires huge reply buffers
  - Usual space for striping information
  - Plus space for "unlink cookies" for client-performed destroy
  - More than doubles space requirements
- There is no reason for older clients to be unable to unlink hugely-striped files
  - So if there is not enough reply buffer space MDS must destroy objects
- MDS-initiated destroys is not free
  - Hogs MDS threads and CPU

Results so far

- Code tested at ORNL 1300 OSTs filesystem
  - Mostly works
- Older clients work fine with updated servers
  - Even when widely-striped files are present
  - Can access files with ~250 stripes
- Metadata operations slower for wide-striped files
  - Not unexpected
- The code is available for interested parties at:
- git://git.whamcloud.com/fs/lustre-dev.git
  - Branch widestriping

whamcloud

Path forward with this simple approach

- Make RPC buffers dynamic
  - No point in penalizing all allocations when only a few files are expected to be widely-striped
  - Make clients retry with bigger buffers is small buffer request failed
- More testing and stability improvements
  - Obviously
- Merging into 2.2 hopefully

Summary

- Brute-force evolutionary approach
  - Limited in scaling
  - Sending megabytes of striping data around is not very practical
  - File attributes scalability is also a problem
    - Hopefully Size-on-MDS is ready soon
- Next, revolutionary approaches we are considering
  - Your input is very valuable

Single FID scheme

- MDS FID to identify all objects
  - No need for precreates anymore. OST objects created on first access
  - Need to ensure objects are not recreated after destroy
- OST index is enough to identify a stripe
  - Reduces per-stripe info to 2-4 bytes
- OST index ranges to further compress striping information
  - 4 (index) + 2 (number) = 6 bytes to store entire range of possibly thousands of subsequent OSTs

Compatibility of single FID and OST range scheme

- Not at all compatible with existing clients
- Possible to unpack this striping on MDS for old clients
  - As long as it fits into provided buffers
  - Puts an extra strain on MDS CPU
- Possible to unpack on Lustre proxies
  - Once we have them
  - Shifts CPU burden from MDS elsewhere
  - Such proxies are still in the somewhat far future

Non-POSIX single FID ideas

- Single file, many individual stripes
  - Each stripe is its own file, cannot be brought together in a concatenated view.
  - Like file per process without the extra metadata overhead
  - Single FID usage brings that to very efficient storage of striping information too

Composite layouts

- Does the client need entire layout
  - Everything we said so far assumed it does
- Complex layouts consisting of multiple bits is also an option
  - Composite layout where each sub layout applies only to some extent in the file
  - Easy to switch in the middle of the file if conditions change (e.g. due to out of space)
  - Client would only request layouts for the range it works with
  - New problems to solve
    - Finding file size, truncate
  - "Joinfile" on steroids

Thank You

- Oleg Drokin

  Senior Engineer

  Whamcloud, Inc.