Lustre Feature Renaissance

Nathan Rutman
Lug 2018
14 Years of Scaling

COMPUTE | STORE | ANALYZE

14 Years of Scaling GB/s

2005 2010 2015 2020
Years of effort spent on performance and scaling

Not an exhaustive list

● ldiskfs scaling, ZFS
● Recovery: interop, VBR, AT, FSCK, Imperative Recovery
● CLIO, MDS rewrite, FIDs
● IO scaling: LRU, read cache, readahead, wide striping, multi-MB rpc, DoM
● MD scaling: statahead, DNE, MMR
● tons of diverse performance improvements
● bugs bugs bugs

Features too of course: mountconf, Kerberos, NRS, HSM, changelogs, pools, multirail
Capable infrastructures in place

- DNE – MD horizontal scaling
- Complex layouts – much more interesting data placement
- FLR – data redundancy inside Lustre

Time to reap some Feature rewards

- Let’s look at some possible features
- These aren’t even designs, just ideas
FLR – data redundancy *inside* Lustre

- One small (?) step for Layouts, one giant leap for Lustre systems design
- No longer need to rely on Failover for data access
- Dual-ported, dual-server, dual-path, dual-$ -$ nope.
- Need:
  - FLR2 = immediate – client writes data durably
  - FLR3 = EC – boo 100% overhead, yay 20% overhead
  - Degraded write support. Track changes for reintegration, or asymmetric layouts?
Spillover Space: death to ENOSPC

- Self-extending PFL (LU-10070, LU-10169)
- Some PFL segments are virtual, instantiated on demand
- Request in a virtual segment requires layout update
- MDS adds a new component on demand
- Can choose the new component striping based on dynamic conditions (e.g. free space)

<table>
<thead>
<tr>
<th>Initial Segment</th>
<th>Extending Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10)</td>
<td>[10,100) 4stripe</td>
</tr>
<tr>
<td></td>
<td>+100,500M ssd</td>
</tr>
<tr>
<td></td>
<td>(,∞] hdd</td>
</tr>
</tbody>
</table>

r/w request past 100:

<table>
<thead>
<tr>
<th>Initial Segment</th>
<th>Extending Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10)</td>
<td>[10,100) 4stripe</td>
</tr>
<tr>
<td></td>
<td>[100,200) ssd</td>
</tr>
<tr>
<td></td>
<td>+100,500M ssd</td>
</tr>
<tr>
<td></td>
<td>(,∞] hdd</td>
</tr>
</tbody>
</table>

r/w request past 300:

<table>
<thead>
<tr>
<th>Initial Segment</th>
<th>Extending Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10)</td>
<td>[10,100) 4stripe</td>
</tr>
<tr>
<td></td>
<td>[100,200) ssd</td>
</tr>
<tr>
<td></td>
<td>[200,300) ssd</td>
</tr>
<tr>
<td></td>
<td>(300,∞] hdd</td>
</tr>
</tbody>
</table>

reach specified max, or insufficient space on ssd
ILM Layouts

- Layout implies an action: stale FLR copy = resync w/ lfs mirror
- And a timeframe: (immediate | eventual)
- Simple ILM policy *already* encoded into layout
- Add some flags to layout and/or policy ref
- Make HSM a true layout (LU-10606): stale HSM copy = resync w/ lfs hsm
- Use Coordinator and Copytool for all movement (LU-6081)

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>8+2 EC HDD</td>
<td>HSM Archive</td>
</tr>
</tbody>
</table>

- □ sync immed
- □ purge after sync
- □ immutable
- □ restore here

- □ sync idle
- □ purge after sync
- □ immutable
- □ restore here

☑ sync immed
☑ purge after sync
☑ immutable
☑ restore here
Asymmetric Layouts

- Reads go to R iff not in W
- Block bitmap on W tracks newly written data
- Client caching and DIO insure full-page writes
- W controls all locks, gives bitmap along with lock grants
- Clients access R or W directly, all under W’s locks

Why?

- Write to flash, read from HDD
- Continue writing to new W if an OST fails (checkpoint) (or ENOSPC)
- EC degraded write case – point of EC is to remain usable in failures

Asym Layout

![Diagram showing Asymmetric Layout with W layer, R layer, and Block bitmap]
Fast Find

- Why do we copy Lustre MD into DB’s or scan raw ldiskfs?
- Need to quickly find files that match certain criteria
- A great ‘lfs find’ could do the same thing, saving the tools effort
  - Server side. RPC from client, returns filtered list
  - Logical combinations of filters
  - Unix-style piping: `lfs find /lustre -size +20M | lfs hsm archive`
- Add new MDT indices to efficiently generate initial candidate lists
  - LRU, file heat, mtime, size
  - `dt_index_operations` (eg IAM) provides generic indexing code
  - Update indices transactionally with MD updates
Rough SoM

- FLR records file size on MDS; comes with sync
- DoM records file size on MDS
- Straightforward to get maximum size, if we don’t care about evicted/failover case
- Rough size is fine for many purposes (e.g. policies)
- Record the quality of SoM, let users decide if usable
- Strict, Rough, Stale, Unknown (LU-9538)
- Don’t return as POSIX size unless strict
Clone Files: extreme create scaling

- File create: ask MDS to create, lock dir, create inode, assign objects
- Clone create: create “all” the files at once
- Single MDS inode, single namespace entry: foo.#
- FID is prefix+#
- Layout is f(FID)

- Shared MD (clones!) but different objects / data / sizes
- open(foo.4,O_CREAT) is now a client-local operation
Alternate Consistency Models

- POSIX API vs POSIX consistency semantics
- Caching allow write coalesce, local latency
- But pay a penalty for locks
- Solutions in Lustre, but requires effort
  - Lockless DIO, Grouplocks
- Make it easier
  - ladvise?
  - Persistent file tags?
  - Automatically change modes?

![Graph showing performance vs transfer size]
All Together Now
Feature Vote?

1. FLR EC with degraded writes
2. Spillover Space **Cray**
3. ILM Layouts
4. Asymmetric Layouts
5. Fast Find
6. Rough SoM **DDN**
7. Clone Files
8. Alternate Consistency Hints
Implementation Plan

1. Ignore Nathan’s slideware
2. <insert smart developer here>
3. Implement!