Lustre & Application I/O

• 2008-04-16
• Oleg Drokin
Agenda

- Lustre overview
- Lustre I/O
- Lustre Tuning
- Jaguar I/O pipeline
- Future improvements
Lustre Architecture

Clustering metadata server grows this pool

Lustre Metadata Servers (MDS)

MDS 1 (active)
MDS 2 (standby)

Lustre Object Storage Servers (OSS)

100’s

OSS 1
OSS 2
OSS 3
OSS 4
OSS 5
OSS 6

Commodity Storage Servers

Shared storage enables failover OSS

Enterprise-Class Storage Arrays & SAN Fabrics

Lustre Clients
10’s - 10,000’s

Router

GigE

Simultaneous support of multiple network types

Elan
Myrinet
InfiniBand

= failover
Why Lustre is fast

• No single point of contention on IO path, data is distributed across many storage servers
• Clients have data caches, could cache dirty data too, immediately know when caches are no longer valid
• Consistent POSIX-compliant view on entire FS from clients without any extra user-performed actions.
Client -> Server I/O in Lustre
Optimal file striping and IO

Clients

File offsets

Object storage
Jaguar I/O pipeline

- Client
- OSS
- Seastar network
  - 7 G/sec
- Fiberchannel
  - 4 Gb/sec
- DDN 9550
  - 300 MB/sec
Normal write path (client)

- A lock is obtained across entire write region (one lock per every stripe affected) - overhead
- Write is split across stripe boundaries and for every stripe we perform:
  - For every new page of data a “grant of space” is consumed for proper OOS detection
  - Data is copied to cache.
  - Pages are put into a batch of dirty pages waiting to grow to RPC size (1M). Per-OST limit of dirty data in cache.
Normal Read path (client)

- For every I/O region a lock is obtained for consistency. One lock per stripe. - overhead
- For every stripe affected by read:
  - Pages are added to a list of pages waiting to grow to RPC size (1M)
  - Readahead can add more pages
  - RPCs are sent (regardless of resulting RPC size)
  - Data is copied from cache to userspace.
Locking (server)

- Lustre locks file data on a per-OST basis with “extent” locks.
- Locks could be of WRITE (exclusive) and READ (shared) type.
- Granularity is page size.
- Attempts to grant as big of a lock as possible without causing any conflicts.
- Notifies other lockholders in case of conflicts to release locks (and dirty data, sync) – expensive!
- No way to change existing locks at the moment
Locking example 1

- client 1 asks for WR [0;4096]
- client 2 asks for WR [4097;8192]
- client 1 asks for WR [8193;12288]
- client 2 asks for WR [0;4096]
- client 2 asks for WR [40960;45056]
- client 2 asks for WR [40960;45056]
- client 1 asks for WR [0;4096]
Locking (client)

- On clients locks are mostly used to guard caches
- Clients cache locks in LRU for later use
- When lock is revoked, all file data in the range is flushed out from that clients cache.
- Locks could be reused between different processes
Read/Write path (server)

- Request is accepted
- Pages are verified to be good (enough space, etc)
- Data is transferred to/from disk directly, bypassing any Linux caches.
- Replies are sent back.
Linux cache & Lustre

- **Dirty cache**
  - 32M (tunable) per OST on every client
  - When there is enough dirty data to fill entire RPC (1Mb), it is being written.
  - Sync or memory pressure can write dirty data in partial RPCs too
  - Lock revocation writes the data and clears the cache

- **Read cache**
  - Memory pressure can flush it
  - Lock revocation flushes it
Direct IO

• No page copying on clients
• No cache
• But still same locking as with normal i/o
• Not all workloads would benefit, consider with care.
MMAP I/O

• Lustre supports mmap that is also fully consistent
• It is much slower for writes.
Important conclusions

• Some local FS guidelines apply
  > Avoid partial page writes

• Lustre-unique hints
  > As few writers per stripe as possible (ideally only 1)
  > Stripe files widely for more throughput
  > Avoid i/o crossing stripe boundaries
  > Submit I/O in as big chunks as possible
  > Avoid repetitive actions from many clients (e.g. truncate shared file to zero from every client)
  > Reading same data from server always hits the disk
System tunables

• Lustre debug can be a performance hog
  > echo 0 >/proc/sys/lnet/debug

• LRU size adjustments (default is 100 locks/cpu)
  > Only makes sense on clients
  > /proc/fs/lustre/ldlm/namespaces/*/lru_size

• Number of RPCs in flight
  > /proc/fs/lustre/osc/*/max_rpcs_in_flight

• Max dirty cache per OST
  > /proc/fs/lustre/osc/*/max_dirty_mb
System tunables

- **Readahead**
  > `/proc/fs/lustre/llite/*/max_read_ahead_mb`
  > `/proc/fs/lustre/llite/*/max_read_ahead_whole_mb`

- **Number of server IO threads**
  > on OSS only
  > module parameter `oss_num_threads` to `ost.ko`
Future lustre improvements

- Less overhead on small i/o
- Cache on Object servers
- Dynamic LRU
- Dynamic number of server threads
- Even better overall scalability
- Bigger RPC sizes
Questions?