Strided Extent Locking

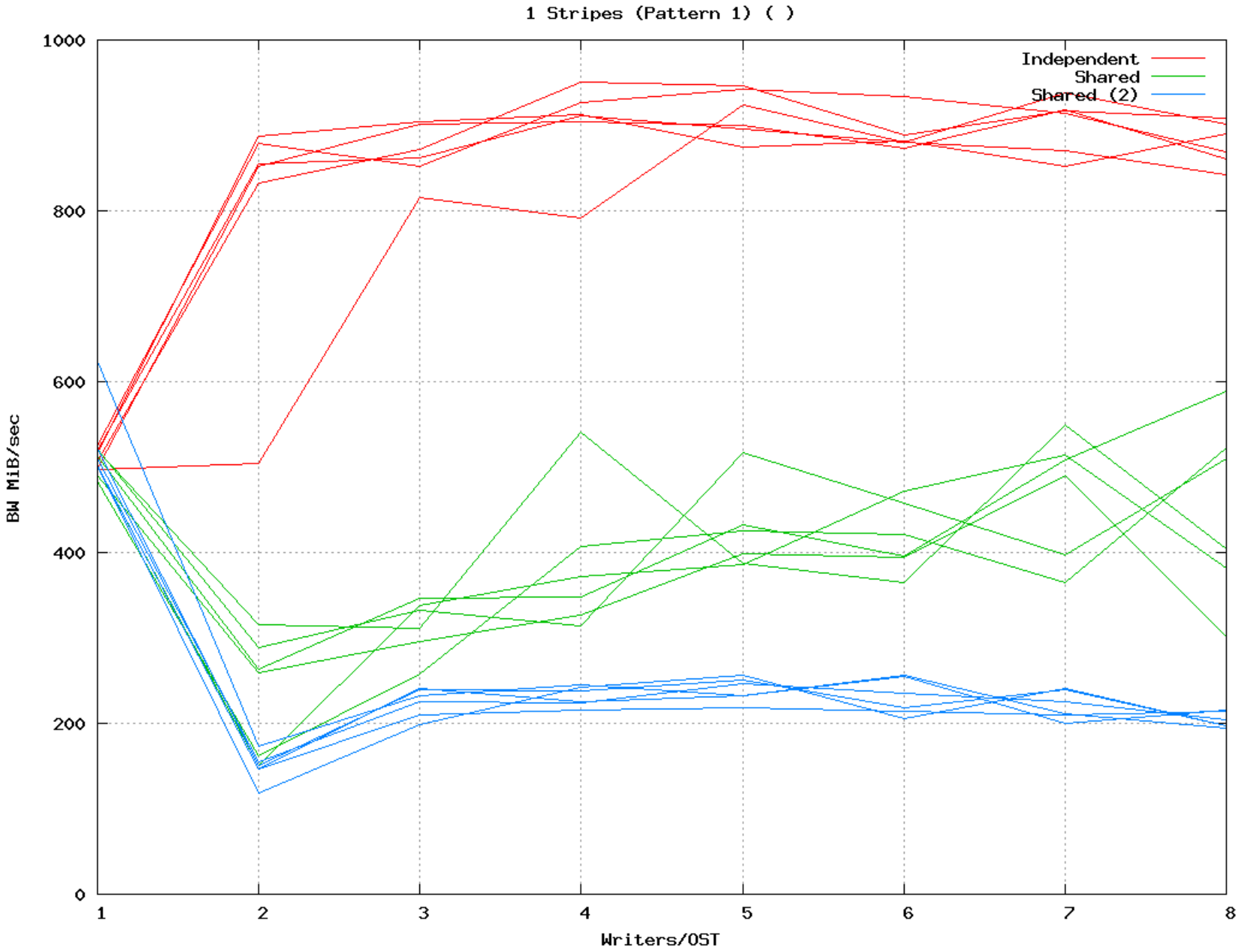
Improving Shared File Performance

Shared File vs FPP

- Two principal approaches to writing out data: File-per-process or single shared file
- File-per-process scales well in Lustre, shared file does not
- File-per-process has problems:
- Heavy metadata load for large jobs
- This isn't getting better: ~250,000 cores on current top 10 x86 machines
- Heavy metadata load during the job and also during post-processing

Shared File Scalability

- Maxes out at one client per OST
- Going from one to two clients reduces bandwidth dramatically, adding more after two doesn't help much
- In real systems, OST can handle full bandwidth of several clients (FPP hits these limits)
- For example, latest Seagate system OSTs have enough bandwidth for 8+ Cray clients per OST


1 Stripes (Pattern 1) ( )
Independent
Shared
Shared (2)
BW MiB/sec
Writers/OST
0
200
400
600
800
1000
1
2
3
4
5
6
7
8

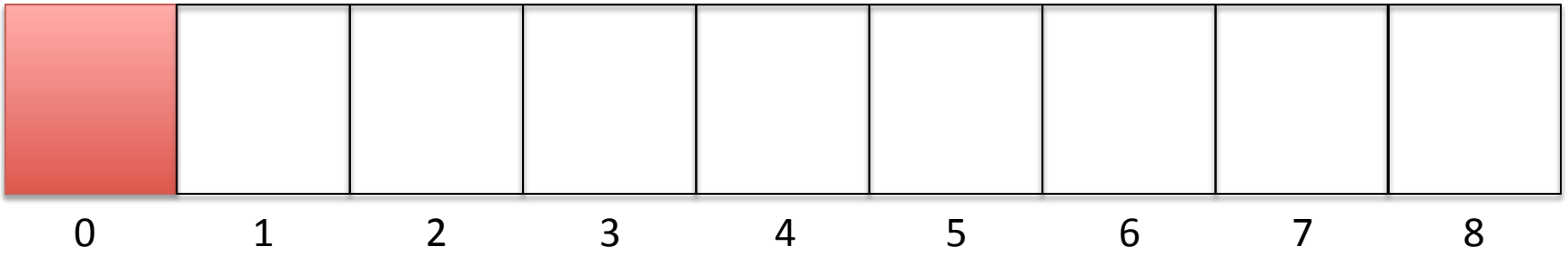
Shared file IO

- Common HPC applications use MPIIO library to do ‘good’ shared file IO
- Technique is called collective buffering
- IO is aggregated to a set of nodes, each of which handles parts of a file
- Writes are strided, non-overlapping
- Example: Client 1 is responsible for writes to block 0, block 2, block 4, etc., client 2 is responsible for block 1, block 3, etc.
- Currently arranged so each client writes to 1 OST

Why doesn't shared file IO scale?

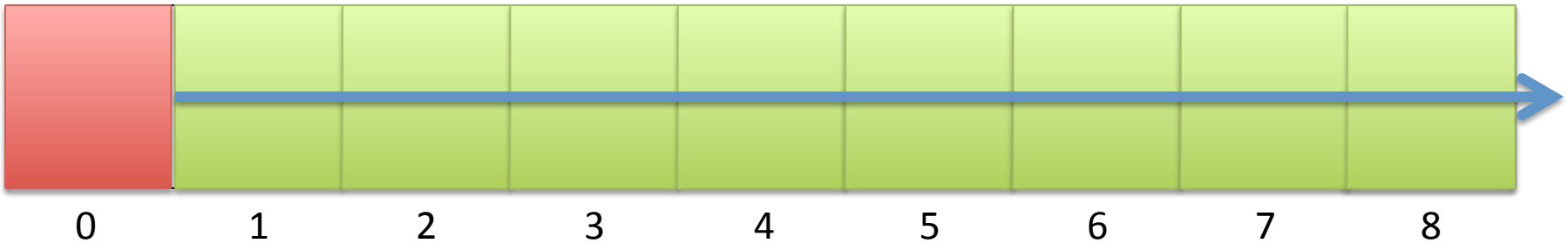
- In 'good' shared file IO, writes are strided, non-overlapping
- Since writes don't overlap, should be possible to have multiple clients per OST without contention
- With > 1 client per OST, writes are serialized due to extent lock design in Lustre
- 2+ clients are slower than one due to lock contention

Extent Lock Contention



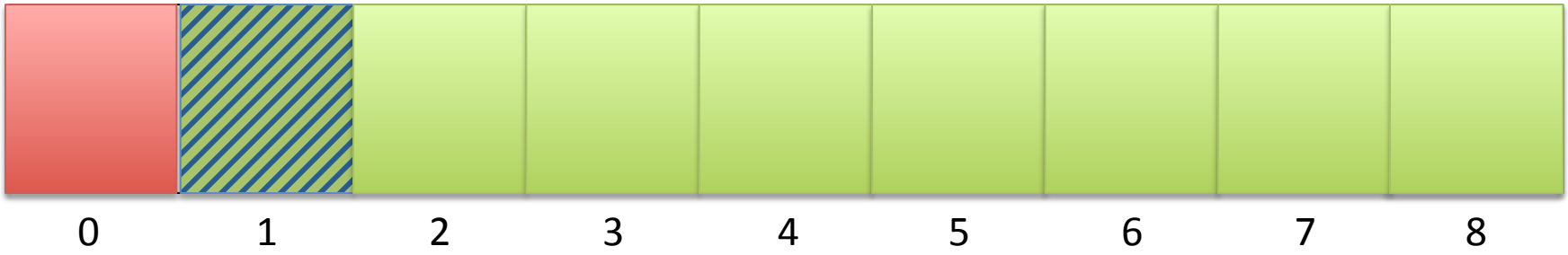
- Single OST view, also applies to individual OSTs in a striped file
- Two clients, doing strided writes
- Client 1 asks to write segment 0 (Assume stripe size segments)

Extent Lock Contention



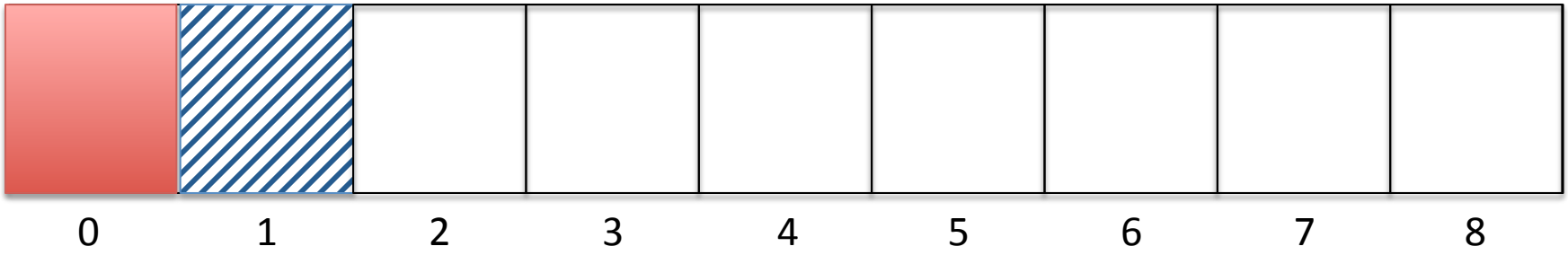
- No locks on file currently
- Server expands lock requested by client 1, grants a lock on the whole file

Extent Lock Contention



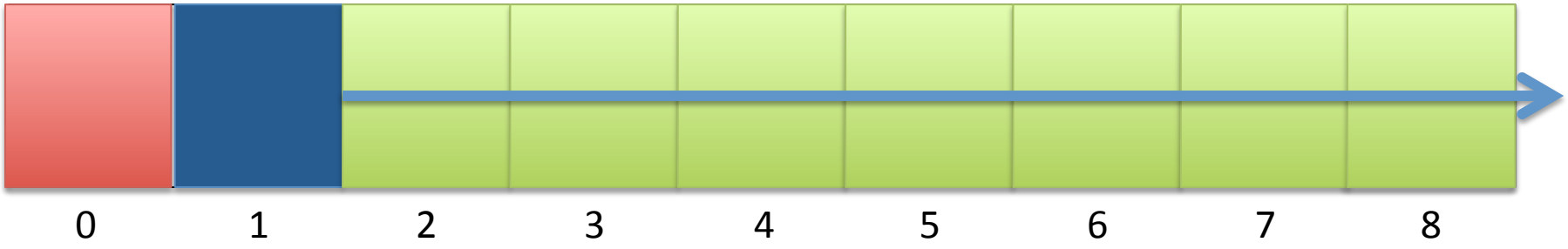
- Client 2 asks to write segment 1
- Conflicts with the expanded lock granted to client 1

Extent Lock Contention



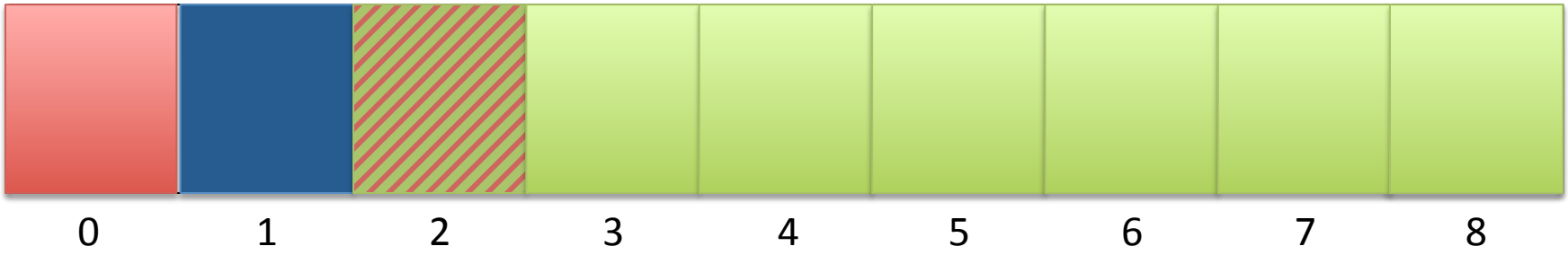
- Lock assigned to client 1 is called back
- Client 2 lock request is processed…

Extent Lock Contention



- No locks on file currently!
- Server expands lock request from client 2
- Grants lock on whole file…

Extent Lock Contention



- Client 1 asks to write segment 2
- Conflicts with the expanded lock granted to client 2
- Etc. Continues throughout IO.

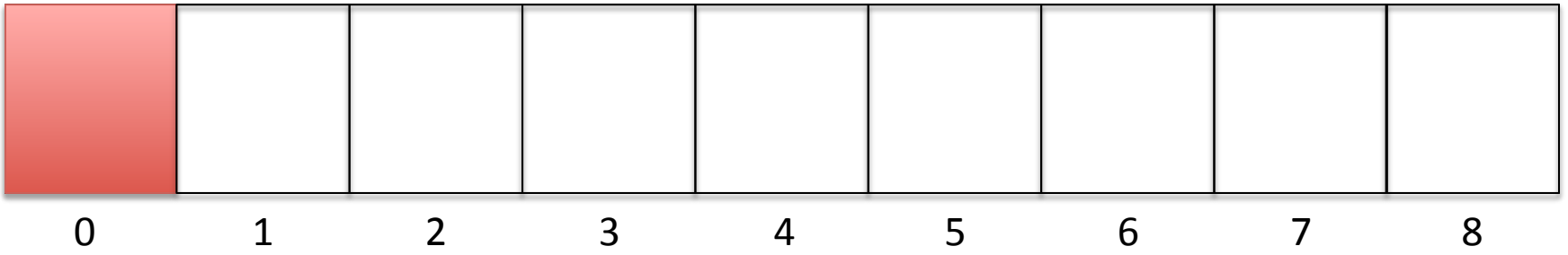
Extent Lock Contention

- Multiple clients per OST are completely serialized
- Even worse: Additional latency to exchange lock
- Mitigation: Clients generally write > 1 segment before giving up lock
- Still slower than one client per OST

Extent Lock Contention

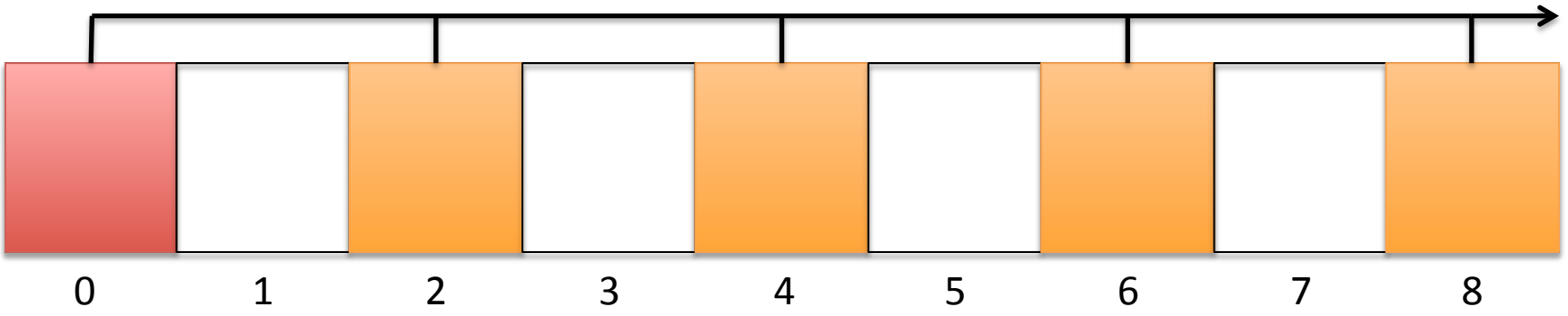
- What about not expanding locks?
- Avoids contention, clients can write in parallel
- Surprise: It's actually **worse**
- Extra latency required to get a lock for every write destroys performance
- That was the blue line at the very bottom of the performance graph…

Strided Locks: Make the lock match the IO pattern



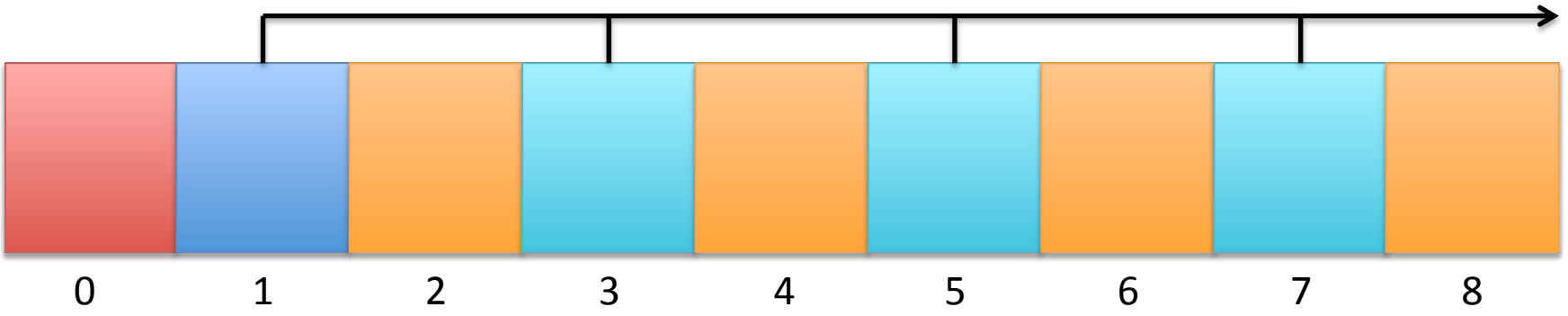
- Imagine a type of lock which matches the IO pattern (All of this is per OST)
- Same situation: Client 1 asks to write segment 0
- Client requests a strided lock (stride of 2) instead of a normal extent lock…

Strided Locks: Make the lock match the IO pattern



- Strided lock with a stride of 2, starting with segment 0
- Lock granted is periodic
- Covers segment 0, 2, 4, etc, to EOF.
- Lock match is defined as a modular function

Strided Locks: Make the lock match the IO pattern



- Next, Client 2 asks for a strided lock with a strided of 2, starting at segment 1
- Covers segment 1, 3, 5,7, etc, to EOF.
- Does NOT conflict with lock for client 1

What about Group Locks?

- Lustre has an existing solution: Group locks
- Basically turns off LDLM locking on a file for group members, allows file-per-process performance for group members
- Tricky: Since lock is shared between clients, there are write visibility issues (Clients assume they are the only one with a lock, do not notice file updates until the lock is released and cancelled)
- Must fsync before releasing the lock & then release the lock to get write visibility between clients

What about Group Locks?

- Works for some workloads, but not OK for many others
- Not really compatible with HDF5 and other such file formats:
  In file metadata updates require write visibility between clients during the IO
- It's possible to fsync and release the lock after every write, but speed benefits are lost

Strided Locks: Performance

- Performance essentially equivalent to file-per-process or group locks
- Simpler to program for than group locks:
  No manual fsyncs
  No write visibility issues
  Doesn't completely lock out other writers/ readers (Normal lock handling applies)

Strided Locks: Performance

- Would like to have a graph here…
- LU-6050: Can't downgrade to 2.5 after running master
- Currently unable to test on large systems at Cray, which are all using 2.5
- Limited testing showed good (FPP equivalent) performance

Strided Locks: MPIIO

- Intended to match up with MPIIO collective buffering feature described earlier, easy to extend to > 1 client per OST
- Freely available in the Lustre ADIO, originally from Argonne, improved by CFS/ Sun
- IOR –a MPIIO –c
- Cray will make a Lustre ADIO patch available

Strided Locks: MPIIO

- Request strided behavior with an ioctl specifying stride count (Writers per OST)
- All writes to this file descriptor create strided lock requests
- Library would use a separate FD for non-strided IO requests
- Hides complexity from codes, they can use MPIIO with collective buffering as before

Strided Locks: Lock size

- Locks are NOT expanded by server
- Currently, strided locks are always given on ‘stripe size’ segments
- Assumes any writes spanning multiple stripe size segments are in error, generates non-strided lock requests instead
- Intended to match up with MPIIO aggregator feature, which splits IO between aggregators in stripe size chunks

Strided Locks: Implementation

- LDLM layer locks: Takes advantage of layering to modify mostly LDLM layer
- Concept of striding exists only in LDLM layer, higher layers of client are unaware
- Stride argument lives in unused bytes of extent version of ldlm_policy_data_t, so doesn't break network protocol
- Will need to add compatibility flag

Strided Locks: Implementation

- Depends on CLIO simplification changes (2.7+)
- Primarily LDLM layer, no significant changes outside it
- A few tweaks to OSC
- Some sticky spots…

Strided Locks: Areas of concern

- Lock matching
- Page management on lock cancellation
- Conflicting extent locks (Interval tree):
  Checking new strided locks against already issued locks
  Checking new locks against strided locks
- Glimpse lock (lvb and ofd_intent_policy)
- Lock longevity & lock time outs
- Lock weighing & known minimum size (KMS)
- Possible issues with replay?

Strided Locks: Lock Matching

- Lock matching is a little tricky, but can be handled with modular arithmetic
- IO which expects a strided lock will not match non-strided locks
- Non-strided IO can use strided locks
- Strided locks only match other strided locks of the same stride (If strides are different, it's a different IO)
- Lock overlap is also manageable

Strided Locks: Page management

- Page management on lock cancellation
- Currently, LDLM layer passes an extent up in to OSC layer and asks that all pages in that extent be handled as appropriate
- Solution: Every time an IO uses a strided lock, add the extent of the IO to a list of used extents in the LDLM lock
- Pass each of these used extents up in to OSC layer using existing machinery
- Implemented in prototype, seems to work

Group Locks: Page Management

- A quick digression…
- The same page management trick might be applicable to group locks
- Group locks currently require an fsync before releasing the lock
- Tracking extents modified under the group lock could potentially remove the need for an fsync
- Easier for user space, possibly a bit faster
- Doesn't solve the write visibility problem

Strided Locks: Conflicting locks

- In some ways, the trickiest part
- Conflicting locks are currently identified using a per-file red-black tree of extents (one tree for each lock mode)
- Sorting property of the tree requires that extents are contiguous, strided locks violate this
- Can’t use the tree for these…

Strided Locks: Conflicting locks

- Use a list, like is done for waiting locks
- Probably a list of granted strided locks
- Consider: A new non-strided lock
- Check it against the extent trees in the normal manner, then against the list of strided locks
- Strided lock count is just writers per OST, so list length should be acceptable

Strided Locks: Conflicting locks

- Consider: A new strided lock
- First check it against the list of strided locks
- For non-strided locks, walk the tree to identify all locks which start after the strided lock
- All of these are **potentially** conflicting
- Check these against the strided lock one-by-one, handle conflicts as they come up

Strided Locks: Conflicting locks

- Strided locks conflict with any strided locks not of the same stride
- Combined with strided locks always using stripe size for the segment size, this makes identifying conflicts easy
- If the first extent of whichever lock starts later in the file does not conflict, the two locks do not conflict
- This is explained in detail in code comments & in forthcoming design doc

Strided Locks: Other concerns

- Lock longevity: Possible timeout issues?
  Group locks are exempt from timeouts
- Replay: Are there issues with replay that aren't covered by changes to LDLM?
- Lock weighing/KMS: The purpose and methods of lock weighing and KMS aren't clear to me
  Seems like list of dirtied extents could be used

Strided Locks: Other concerns

- Glimpse lock and ofd_intent_policy are murky as well, not sure how or if they need to change
- Testing: Need to work up a thorough test suite for these
- Minor: Getting stripe dimensions in to LDLM layer for lock request, minor layering violation (and where to put it?)

Questions

- Have I missed a major reason this idea is unworkable?
- Comments on my areas of concern: Page management, lock contention, lock longevity, replay, lock weighing?
- Specific issues with the design as presented?

Other Information

- Detailed design doc will be forthcoming, explains math behind matching & overlap
- Prototype code, a patch against master, is here: **LU-6148**
  Very much a prototype, mainly for design discussion (Not even fully implemented yet)
- Thank you to Cray engineers David Knaak Bob Cernohous for the idea and assistance testing