Lustre & Small I/O:

Size does matter (Unfortunately)

Small I/O

- Very hard to offer good performance for small I/O
- 'Small' is anything less than various natural boundaries – RPC size is a notable one
- The smaller the I/O, the worse the performance
- Natural minimum I/O size is 1 page, anything smaller can be especially bad

Why is it so bad?

- Client side per I/O overhead
  - Much worse on Lustre than local fileystems
  - Lots of work done regardless of I/O size
  - Locking, cache management, etc, really adds up
- No obvious pain points – Death by a thousand cuts
- Network costs per I/O
- Disk hardware limits (small I/Os terrible for spinning disk, not good for flash)

What do we do for small I/O now?

- Re-use LDLM locks (most I/Os already have required lock)
- Sequential:
  - Read ahead and write aggregation
    - Avoid small I/Os over network/to disk
    - Still have to process small I/Os on client
- Random
  - Tell people “Please don't do that.”
  - Direct I/O (Lower locking overhead)

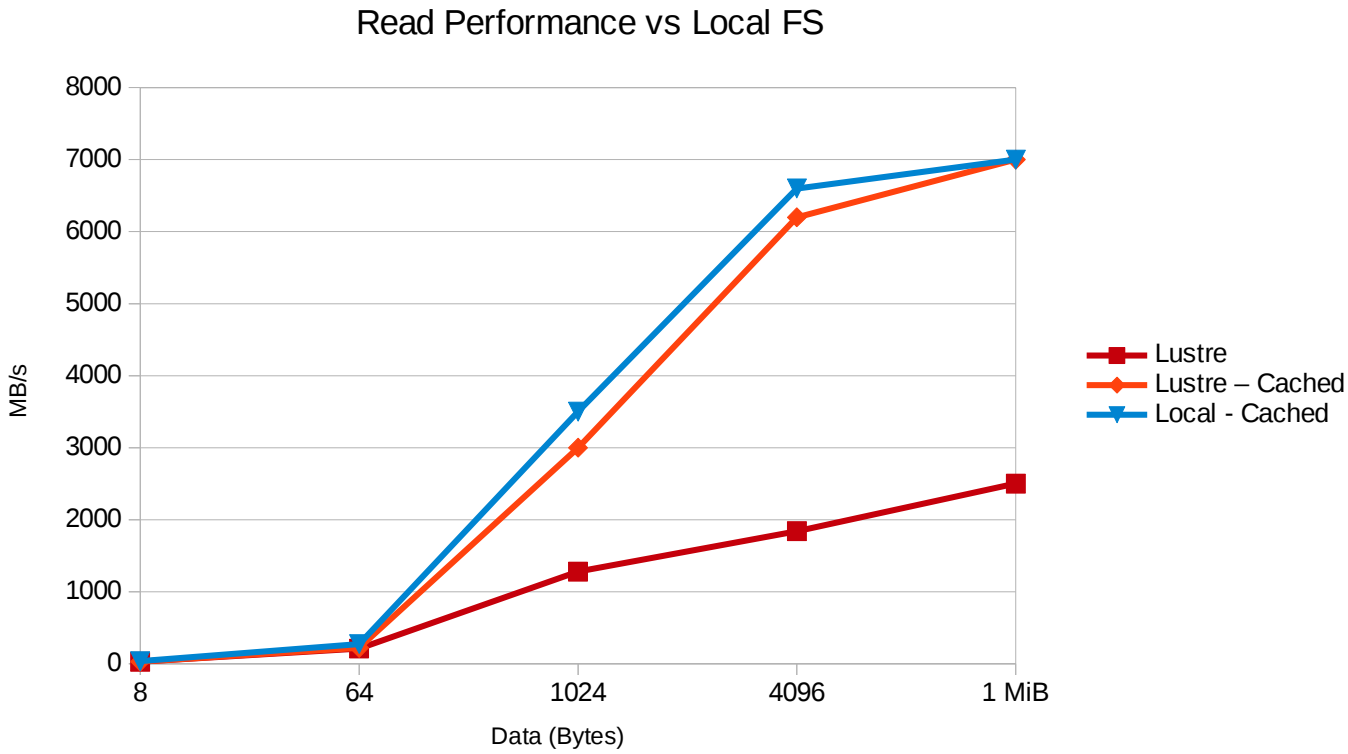
Reads

- Readahead: Read more data than asked for
  - Guarantees large I/O
  - Could be better if more asynchronous (Tough, though: See LU-8964)
- Per I/O overhead still bad for small reads
  - ‘Fast Reads’ - Really clever idea, Jinshan Xiong (Intel)

Fast Reads

- Read overhead is mostly to guarantee pages are present & verify locking
- But if a page is present and up to date, it must be locked correctly
- So just read pages directly from page cache (minimal interaction with Lustre)
- Really, really fast. Improves large & small I/O.
- Still a bit more overhead to squeeze out: LU-9749 (landed)

Read Performance vs I/O Size



Writes

- Writes are harder – Pages are usually created by writing, so not already present
- Must update file size
- Out of space (grant) issues
- OSC layer must know about dirty pages for writeout
- If a dirty page is present, we know this is handled already. Can we use that...?

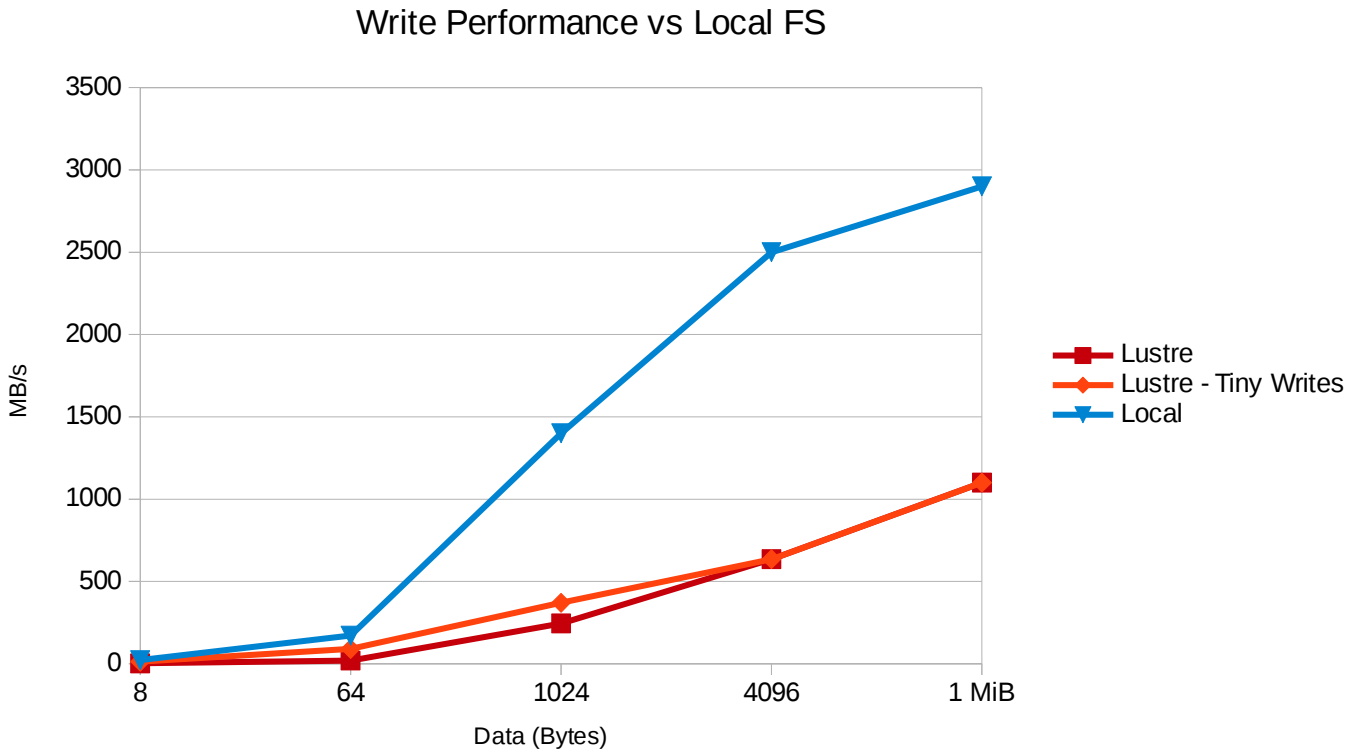
Tiny Writes

- Except for really small (< 1 page) sequential writes
- If writing a few bytes at a time, dirty page will usually be present
- Hence, tiny writes:
  When a write is < 1 page in size and page is already dirty, write directly to that page without cl_io
- Have to update file size, HSM dirty state
- LU-9409 – Not landed yet.

Write Performance vs I/O Size

Bytes	Lustre	Lustre + Tiny Writes	Local
8	2.3 MB/s	12 MB/s	22 MB/s
64	19 MB/s	90 MB/s	171 MB/s
1024	245 MB/s	370 MB/s	1400 MB/s
4096	635 MB/s	635 MB/s	2500 MB/s
1 MiB	1100 MB/s	1100 MB/s	2900 MB/s

Write Performance vs I/O Size



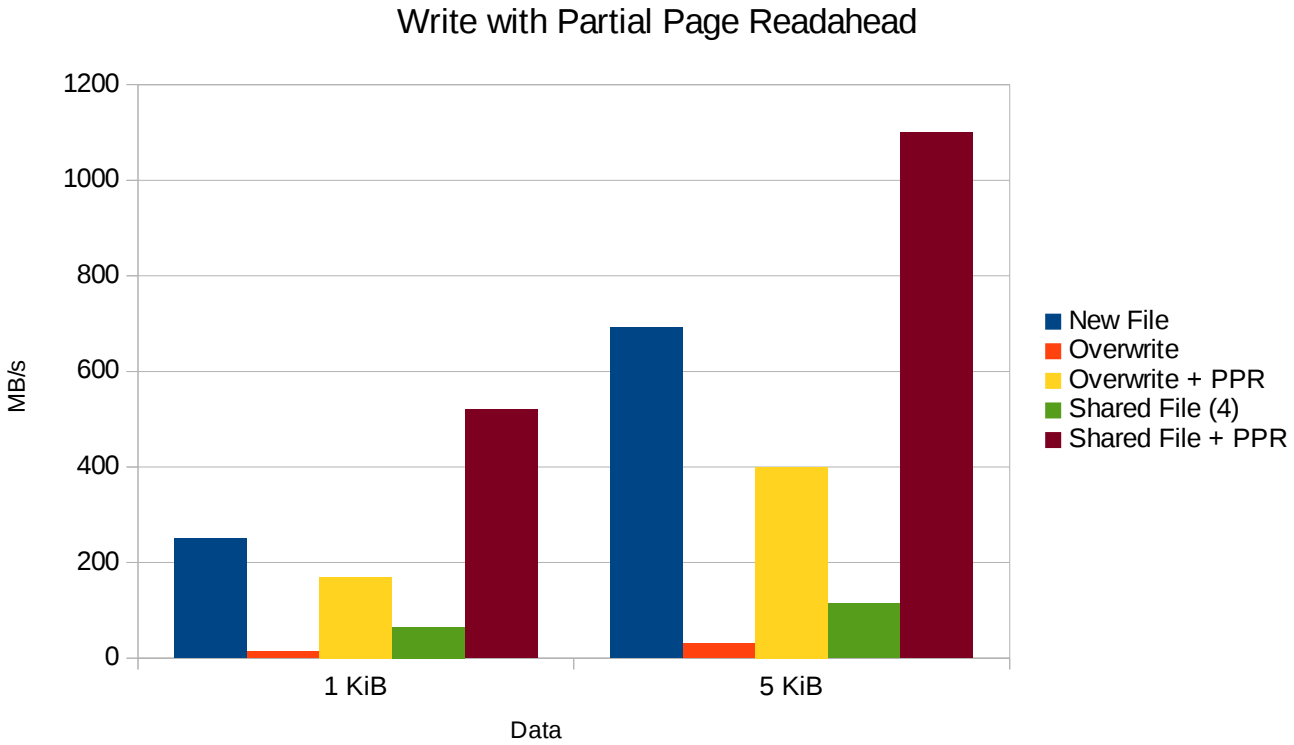
Partial Page Writes 1: Readahead

- Overwriting a file at small sizes is painful
- Have to read in each page before writing it
- Shared file writing also counts as overwriting – can't know pages are empty
- Read in one page at a time... Very slow.
- Use readahead!
- LU-9618: Partial page readahead (PPR, Patrick F./Jinshan) – Not landed yet

Write Performance with PPR

Bytes	New file	Overwrite	Overwrite + PPR	Shared file (4 Writers)	Shared file+PPR
1KiB	250 MB/s	13 MB/s	170 MB/s	65 MB/s	520 MB/s
5KiB	692 MB/s	30 MB/s	400 MB/s	114 MB/s	1100 MB/s

Write Performance with PPR



Partial Page Writes 2: Extent Awareness

- Shared file writing of a new file could be better still
- Not overwriting, so no data in those pages – we know this, but Lustre/VFS doesn't
- Rough sketch:
  Client tracks size reported by server when write lock granted
  Pages > than that size that this client didn't write haven't been written to & don't need to be read
  Could use osc_extent to track this, extents would have to live as long as their covering LDLM write lock (not only as long as their underlying pages – pages could be evicted)

Partial Page Writes 2: Extent Awareness

- Mostly applicable to single client shared file (but would help some for multiple clients)
- Rough calculations (comparing to 4K shared file writes) suggest ~20% benefit for 5K
- Expect up to double that for 2K and smaller
- Probably not worth the time. Maybe some day.

Write Containers

- Tiny writes are very limited in applicability, can we do better?
- Write containers (Jinshan Xiong, Intel)
- Prepare many per I/O items in advance and/or do them in a batch (Ex.: Locking, grant, dirty page tracking)
- Design stage only, Jinshan is looking for volunteers
- Expect improvements of several times for smaller I/O
- Reduced contention for shared file I/O
- Only benefits sequential I/O, adds complexity

Small Random I/O

- Can't do readahead
- Can't batch at all to disk
- Yuck.
- We do batch writes at RPC layer, benefit is significant
- Flash on servers helps a lot here (Spinning disk random IOPs are... bad.)

It's all about Latency

- If you can't batch I/O, then do it as fast as possible
- Lustre latency is still death by a thousand cuts, but some things help
- Direct I/O is slightly better than buffered I/O (less locking)
- LU-1757 – Immediate short I/O (Alex Boyko, refreshed by Patrick F.) [Not landed yet...]

LU-1757: Immediate Short I/O

- RPC required to set up RDMA for bulk transfer
- For small transfers, extra round trip is worse than larger non-RDMA message
- Ergo, put small I/Os in to buffer in RPC
- Straightforward, but limited benefit
- About 30% on 4K reads on Cray Aries to flash (Slower network would give a larger benefit)
- Too small to measure on writes (Most time spent in journaling)

Summary

- Small I/O stinks. Random small I/O really stinks.
- Sequential: Reads are good, writes are bad
  Tiny writes (LU-9409)
  Partial page readahead (LU-9618)
  Write Containers
- Random:
  Immediate short I/O (LU-1757)

What next?

- **Sequential:**
  Review & landing existing patches
  Write Containers
  Async readahead
- **Random writes:**
  Journaling – Can we make this faster?

One more thing...

- Conflicting small I/O is particularly horrible, LDLM latency
- Small improvement possible:
  LU-4198: Lockless direct I/O
- Client LDLM locking not strictly necessary for direct I/O, since there are no pages to protect
- Not much use in non-conflicting case
- LU-247: Unaligned direct I/O (very old, complex) – Could improve the range of workloads benefiting from LU-4198