Small I/O Performance Improvements: Pleasant & Unpleasant Surprises

LUG 2018

Small I/O

- Distinct problem from small files (though commonly found together)
- Very hard to offer good performance for small I/O
- 'Small' varies by who you ask: less than various natural boundaries (page size, RPC size, etc)
- The smaller the I/O, the worse the performance
- Natural minimum I/O size is 1 page

Unpleasant Surprises

- Crossing some size boundaries leads to nasty surprises
- Unaligned write to existing files can be 95% slower
- I/O < 1 page in size gets worse & worse, even though Linux does I/O 1 page at a time
- Poor user experience - “4096 bytes was fine, why is 4097 bytes terrible?”

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
- Network costs per I/O
- No obvious pain points – Death by a thousand cuts
- 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
  - Unaligned Overwrites
  - ‘Fast Reads’ - Andrew Perepechko (Cray), Jinshan Xiong (Uber)

Surprise #1: Unaligned Overwrites

- **Overwriting an existing file is the same as a new write, until it's suddenly not**
- **I/O happens a page at a time, must read in partial pages**

Bytes	New File	Overwrite	Overwrite/ New File
4096 (4K)	600 MB/s	600 MB/s	100%
4097	590 MB/s	**18 MB/s**	**3%**
8192 (8K)	900 MB/s	900 MB/s	100%
8193	880 MB/s	**35 MB/s**	**4%**

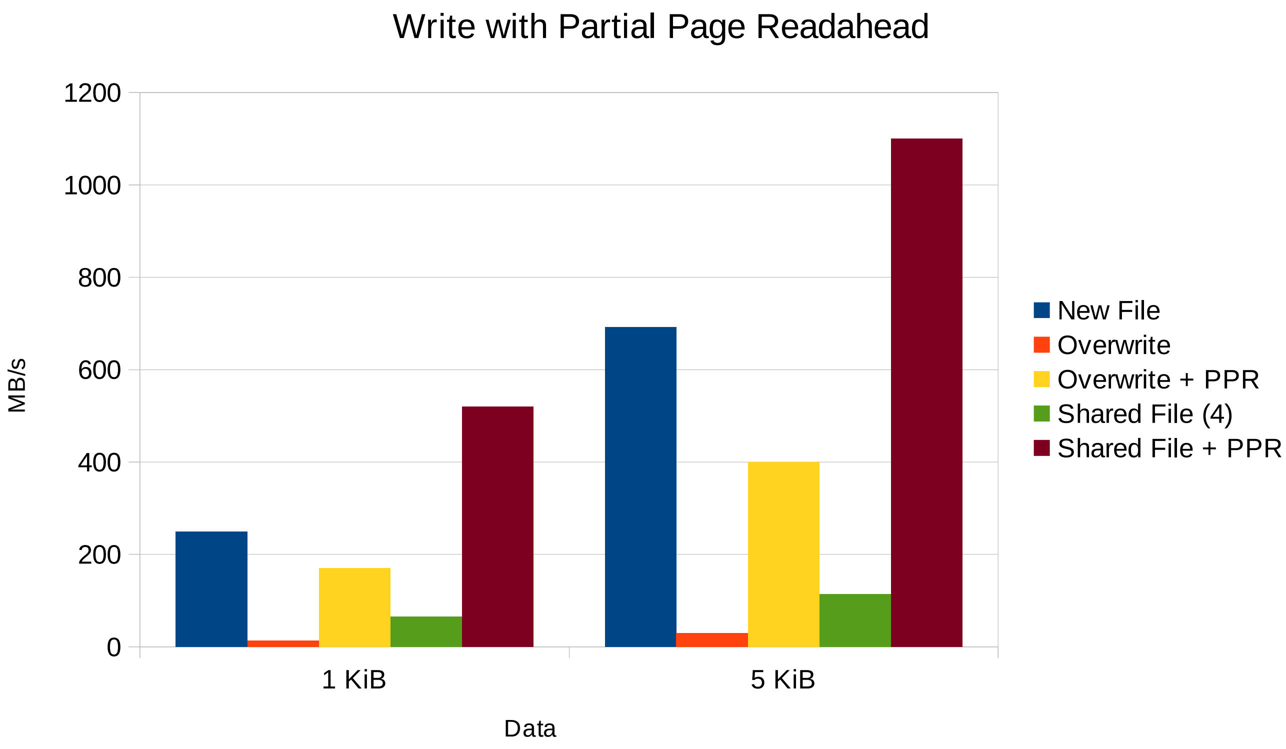
Partial Page Readahead

- Shared file writing also counts as overwriting – can't know pages are empty
- Read in one page at a time... Very slow.
- We have a solution for this: Use readahead!
- LU-9618: Partial page readahead (PPR, Patrick Farrell/Jinshan Xiong)

Write Performance with PPR

Bytes	New File	Overwrite	Overwrite/ New File
4096 (4K)	600 MB/s	600 MB/s	100%
4097	590 MB/s	**401 MB/s**	**70%**
8192 (8K)	900 MB/s	900 MB/s	100%
8193	880 MB/s	**598 MB/s**	**68%**

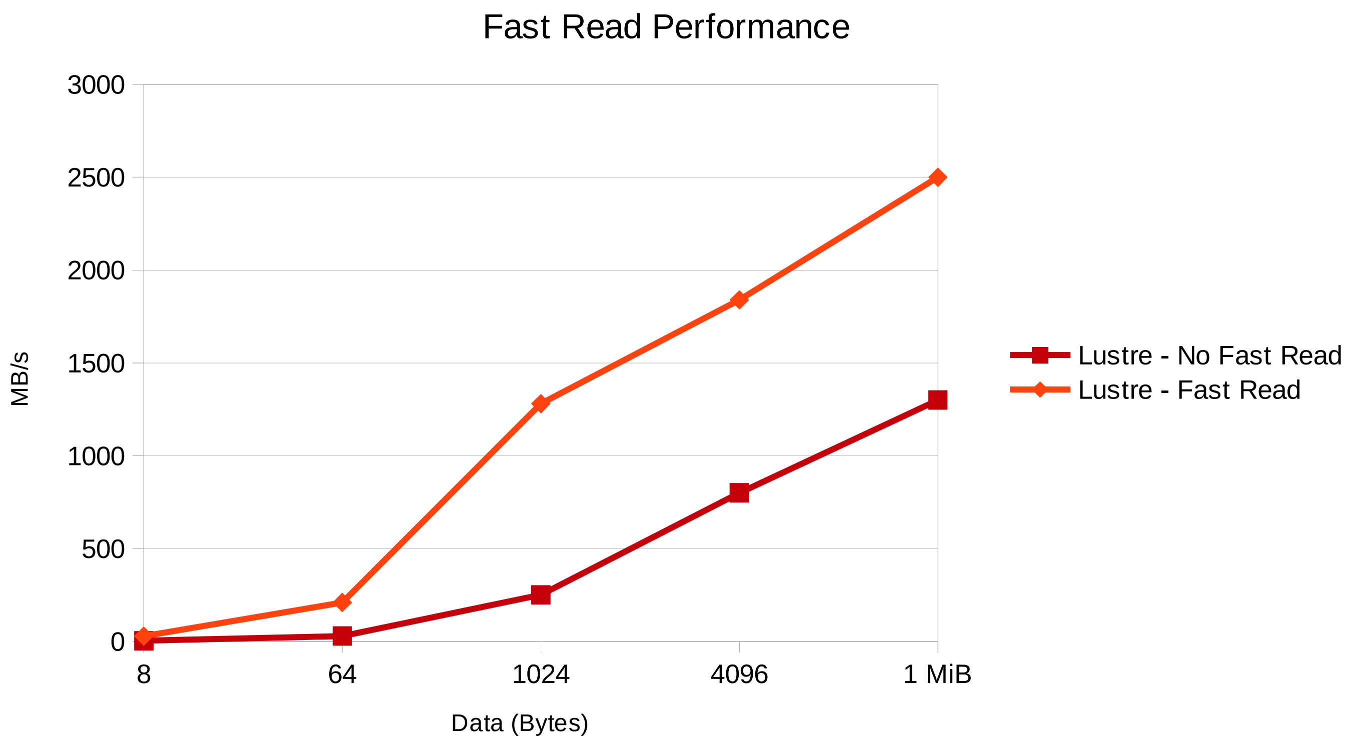
Write Performance with PPR



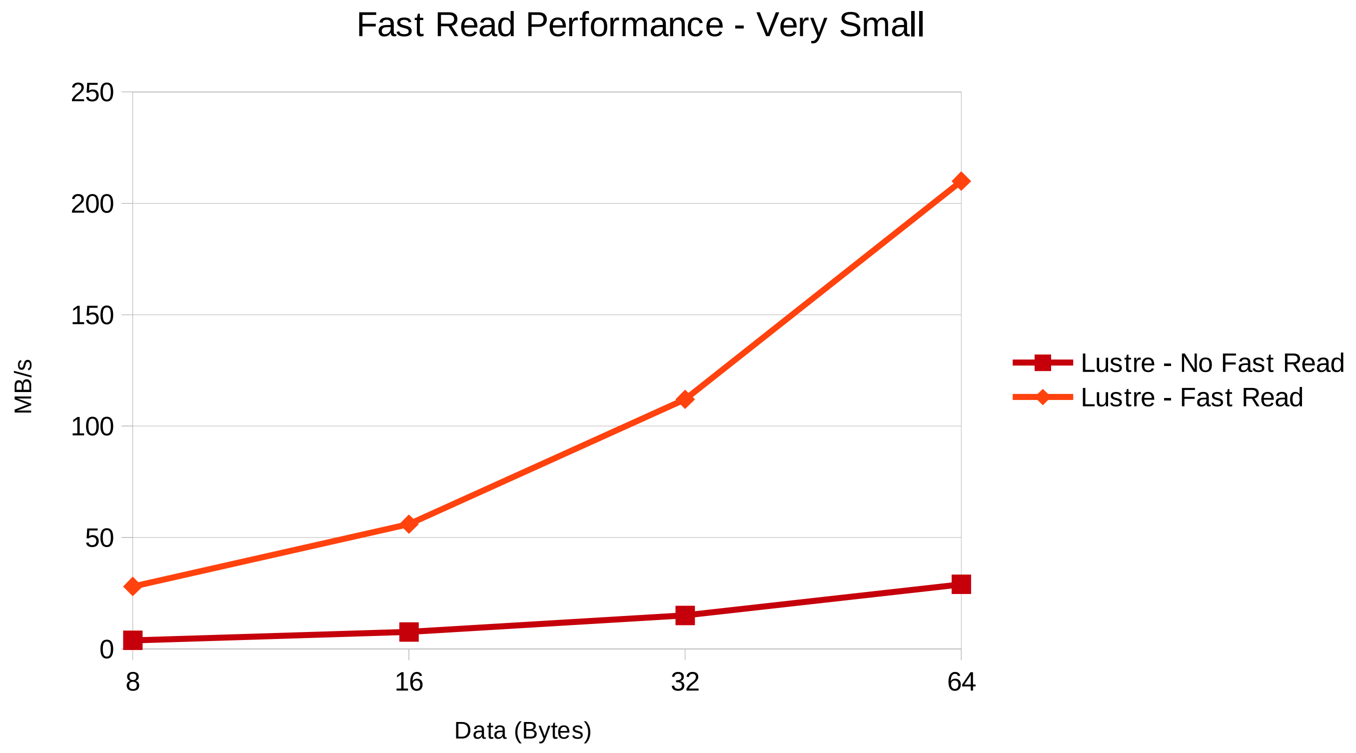
Fast Reads

- Readahead brings in pages before they’re needed
- So, most userspace reads are satisfied from cache
- Old read code does a lot of work to check locking for cached pages
- But LDLM evicts pages on conflicting writes, so we can assume all cached pages are safe to read
- Really, really fast. Improves large & small I/O.
- Landed in 2.7-2.8 time fame

Read Performance vs I/O Size



Read Performance vs I/O Size



What about writes?

- Writes are harder – Pages are usually created by writing, so not already present
- More complicated than reads:

  File size, ENOSPC (grant) handling, dirty page writeout.
- If a dirty page is present, we know (most of...) this is handled already. But so what? Dirty pages aren't present until we write to them.

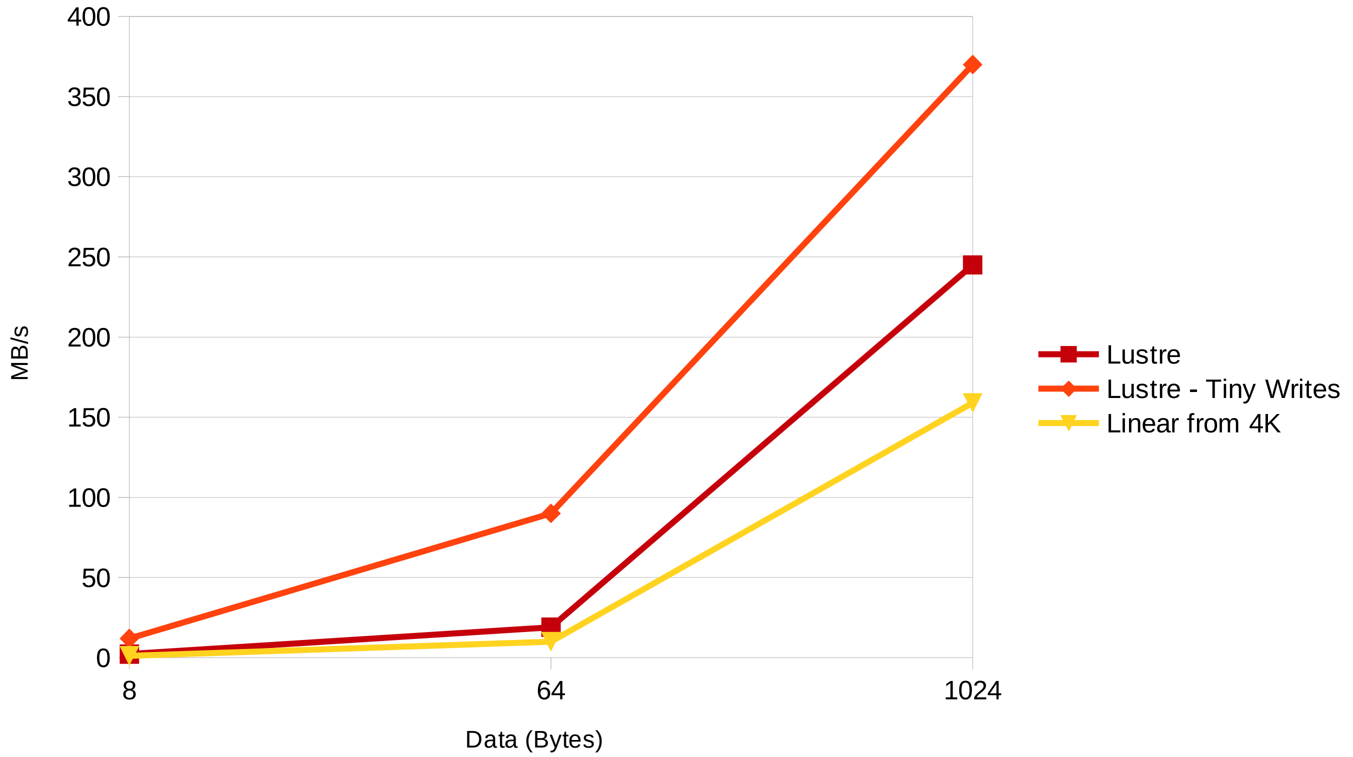
Surprise #2: Tiny Writes

- 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 full i/o
- New feature in 2.11

Write Performance vs I/O Size

Bytes	Lustre	Lustre - Linear	Lustre + Tiny Writes
8	2.3 MB/s	1.2 MB/s	12 MB/s
64	19 MB/s	10 MB/s	90 MB/s
1024	245 MB/s	159 MB/s	370 MB/s
4096	635 MB/s	635 MB/s	635 MB/s

Write Performance vs I/O Size



Possible Future: Write Containers

- Tiny writes are very limited in applicability, can we do better?
- Write containers (Jinshan Xiong)
- Prepare many per I/O items in advance/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
- We do batch writes at RPC layer, benefit is significant
- Flash on servers helps a lot here (Much better IOPs than spinning disk.)

It's all about Latency

- If you can't batch I/O, then do it as fast as possible
- No silver bullets
- Direct I/O is slightly better than buffered I/O (less locking)
- Network request latency (smaller on HPC networks, but still matters)

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
- About 30% faster 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 is hard, especially for a parallel file system
- Lustre 2.11 contains some significant improvements
- Sequential: Reads are good, writes are OK
  Tiny writes (LU-9409)
  Partial page readahead (LU-9618)
  Write Containers
- Random:
  Immediate short I/O (LU-1757)

What next?

- Sequential:
  Tiny write append
  Write Containers
  Async readahead
- Random writes:
  Journaling – Can we make this faster? Special “no journal” mode for non-critical data?