Parallel-readahead: a new readahead framework for Lustre

Li Xi, Shuichi Ihara

DataDirect Networks

Motivation

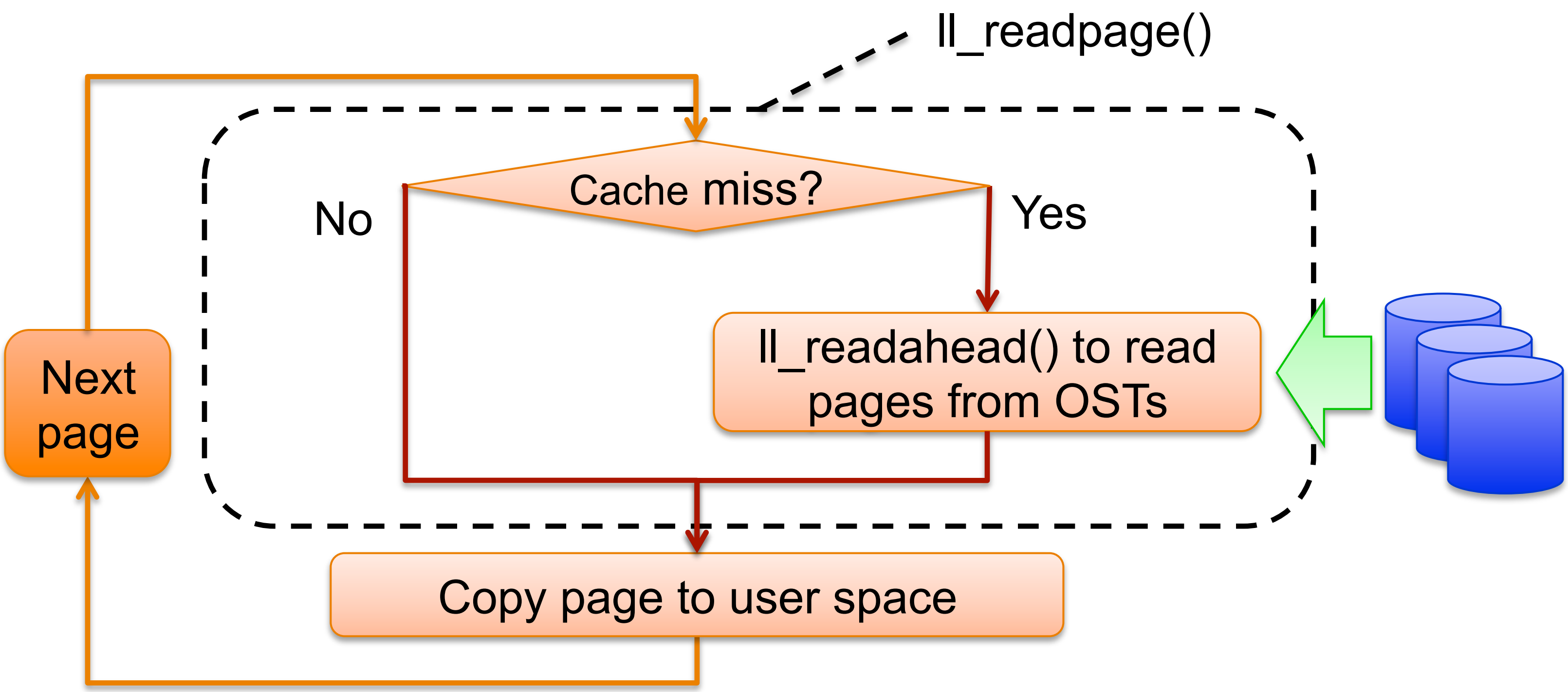
- **Single thread I/O performance is important**
  - Single thread I/O is common even in parallel applications
  - The time cost of single thread I/O in parallel applications can't be reduced by adding compute nodes
  - The benefit of changing single thread I/O to multi-thread/distributed I/O might not worth the effort because I/O is on the single file
- **Good readahead algorithm is critical for read performance especially for single thread read**
  - Latency of non-cached I/O is still high because RPC is still expensive
- **Single thread I/O performance of Lustre is much slower than the client's total bandwidth**
  - The fast growing bandwidth of network and storage is enlarging the gap

Background

- **Hardware specification has been improved a lot**
  - Memory size on client is becoming larger thus could support readahead algorithm that is more aggressive
  - CPU frequency stands still, but number of CPU cores keeps on growing, so CPU cost is critical for single thread read
- **Software improvements enable aggressive readahead**
  - Page cache management of Lustre client has been updated from private management (Lustre-1.x) to Linux kernel (Lustre-2.x)
  - Page cache management of Linux kernel is efficient and smart enough to support aggressive readahead
  - Fast read patches (LU-8149) reduce latency of cached read, thus readahead has become even more important to read performance

Current readahead in Lustre

- Current readahead algorithm dates back to 2004 (Lustre-1.2 or ealier) and has been updated from time to time but not replaced by new ones

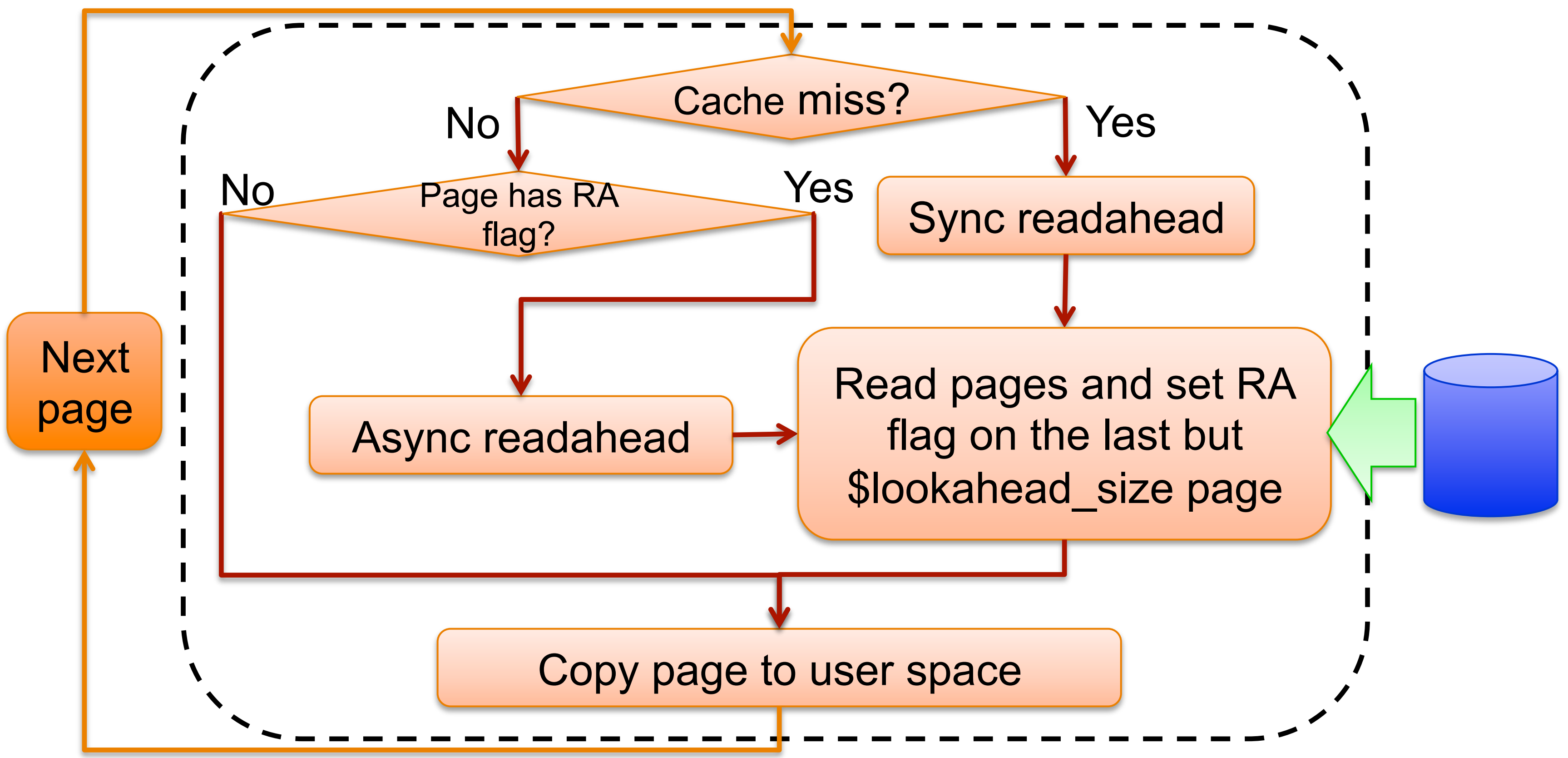


Why current readahead needs improvement?

- **An estimation of the read speed**
  - 1/speed = (1 - cache_miss_rate) / cache_speed + cache_miss_rate / none_cache_speed
  - cache_speed has been improved a lot by fast read patches
- **Cache miss rate of current Lustre readahead is high**
  - A good readahead algorithm would reduce cache miss rate to zero
  - Readahead of Lustre will only be triggered when cache miss, thus cache miss rate will always lager than (size_per_read / size_per_readahead)
- **Lustre readahead window has some problems**
  - Even readahead window size is large, most of the window is behind the accessing offset
  - Readahead is not able to fill the OSC RPC slots
  - The bandwidths of OSCs are not fully used even under heavy I/O of single thread
- **Codes of Lustre readahead are complex thus hard to tune or improve**
  - Status of sequential mode and stride mode are mixed together

Ondemand readahead of Linux kernel

- Ondemand readahead is the current algorithm used by Linux kernel since Linux-2.6.20



Ondemand readahead of Linux kernel

Readahead happens in two cases

- Synchronous readahead happens on cache miss
  - I/O will happen anyway, so synchronous readahead reads more pages together in a single I/O
- Asynchronous readahead happens on lookahead page
  - The prefetched pages should be at least lookahead_size ahead of current access offset.
  - When the page with PG_readahead(RA) flag is being accessed, the prefetched pages in the front are dropping to lookahead_size.
  - In order to avoid future cache miss, do readahead when page with RA flag is being accessed

Ondemand readahead on Lustre

- **We ported ondemand readahead algorithm to Lustre**
  - Max readahead window is increased from 2MB to 40MB
  - Single thread read performance with old readahead is about1.0 GB/s
  - Single thread read performance with ondemand readahead is about 1.4GB/s, 40% improvement
- **But ondemand readahead is still not enough for Lustre**
  - It was designed/optimized for local file systems
  - Its maximum readahead window size is too small (<2MB)
  - It doesn’t detect stride read
  - It doesn't always try to prefetch in large IO
  - It is not aware of Lustre stripe and LDLM lock

Design of Parallel-readahead

Multi-thread prefetch

- Parallel prefetch: CPU speed limits the read performance if all readahead is done by the read process
- Real asynchronous prefetch: Asynchronous readahead is done in a thread pool rather than the read process

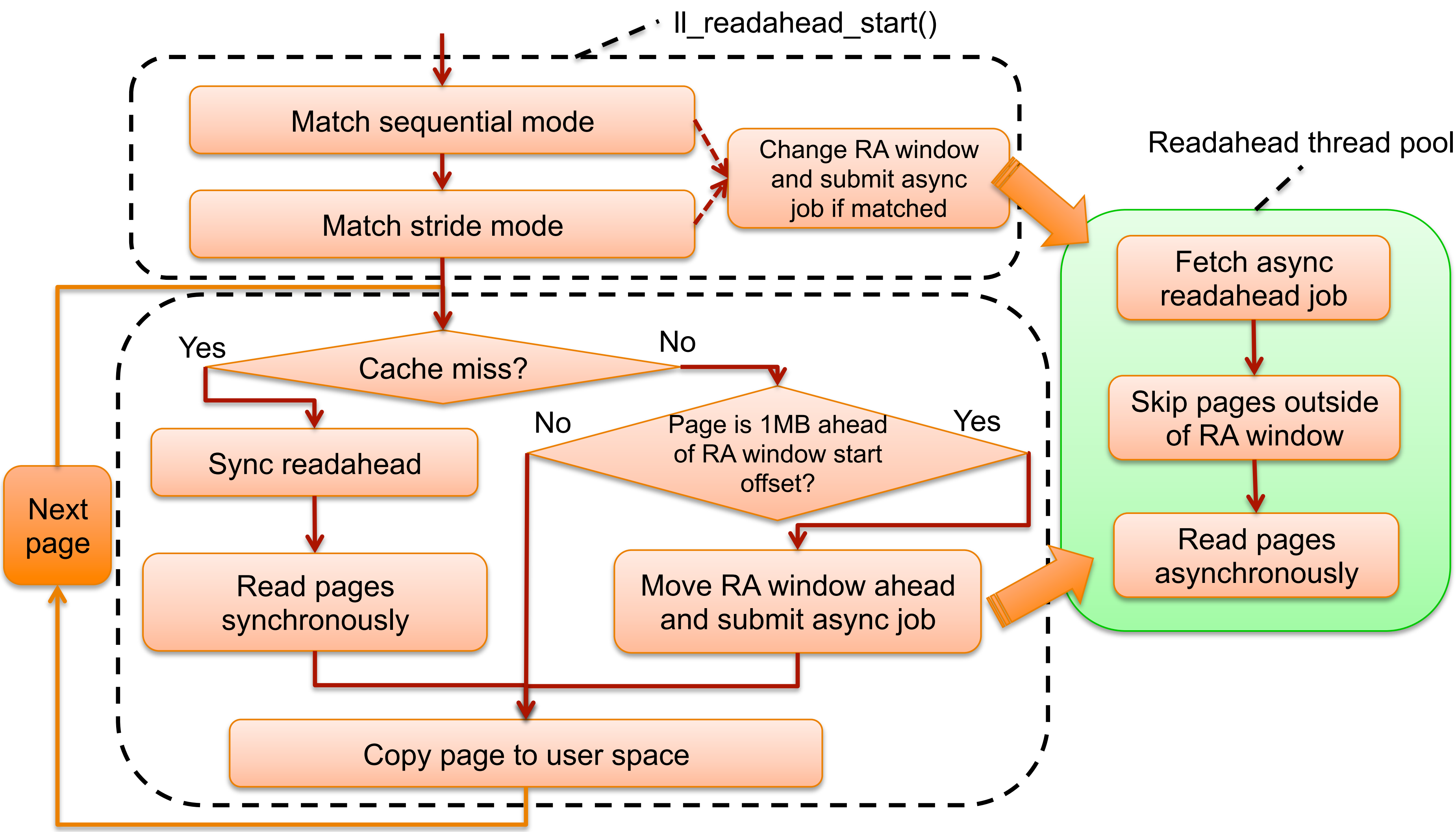
Readahead trigger timing

- Synchronous readahead is done in the read process when cache miss happens since the page is being waited
- Asynchronous readahead is triggered at the very beginning of read syscall for time saving
- Asynchronous readahead is also triggered when read syscall makes large progess

Pattern detection

- Both sequential read and stride read are detected and speeded up
- The framework is extendable so that pattern detection and readahead policies could be added in the future for patterns such as random read, semisequential read, backward read, interleaved read, etc.

Framework of Parallel-readahead

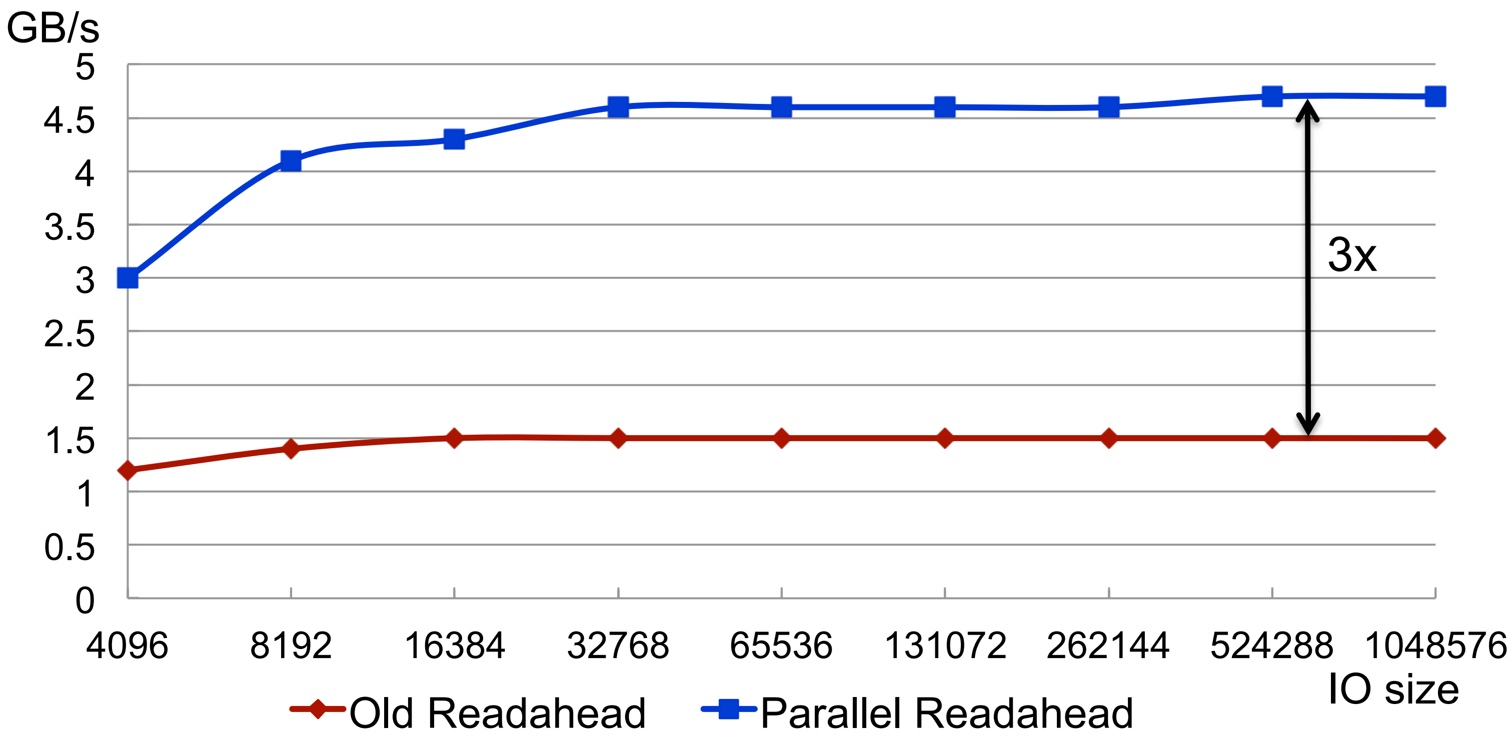


Benchmark configuration

- **All performance results are done in a Lustre file system with all servers and client running on a single server**
  - 64 GB memory
  - 500 GB SSD(SAMSUNG 850 EVO 500G SATA3)
- **The SSD can only provide bandwidth of about 545 MB/s**
- **Apply a patch that bypasses read on OSD (LU-7655)**
  - Lustre client could get over 4.5 GB/s read (fake) bandwidth
  - This should be an environment which is very simple but suitable for running readahead benchmarks
- **Lustre version: IEEL3 on CentOS7**
  - Fast read patches are included
  - CentOS7 can get more performance than CentOS6, 4.5 GB/s VS 3.3 GB/s

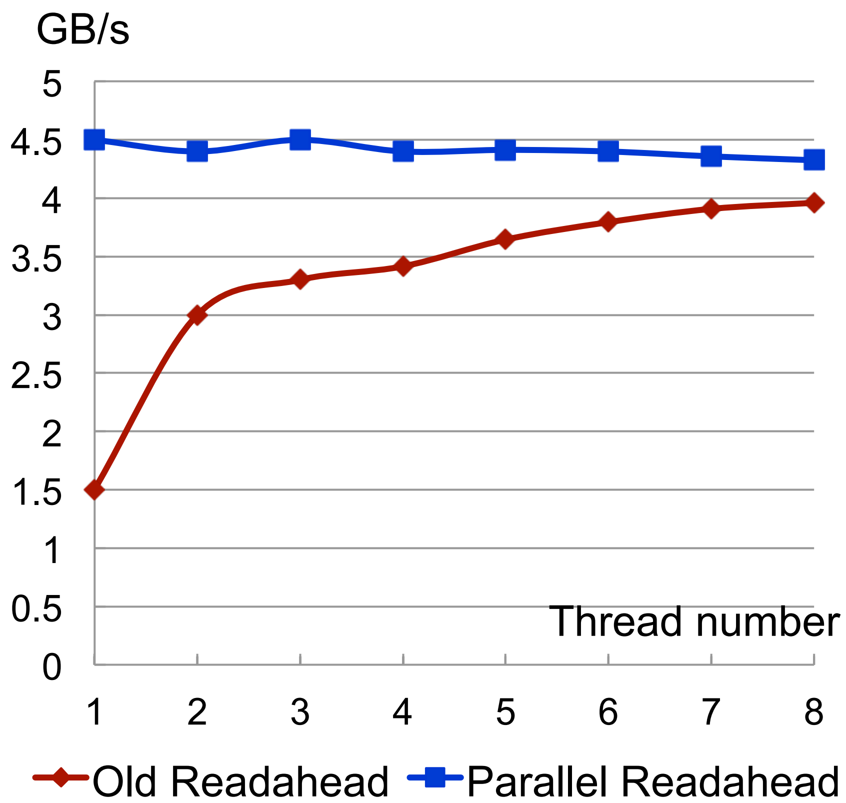
Benchmarks: Single thread read

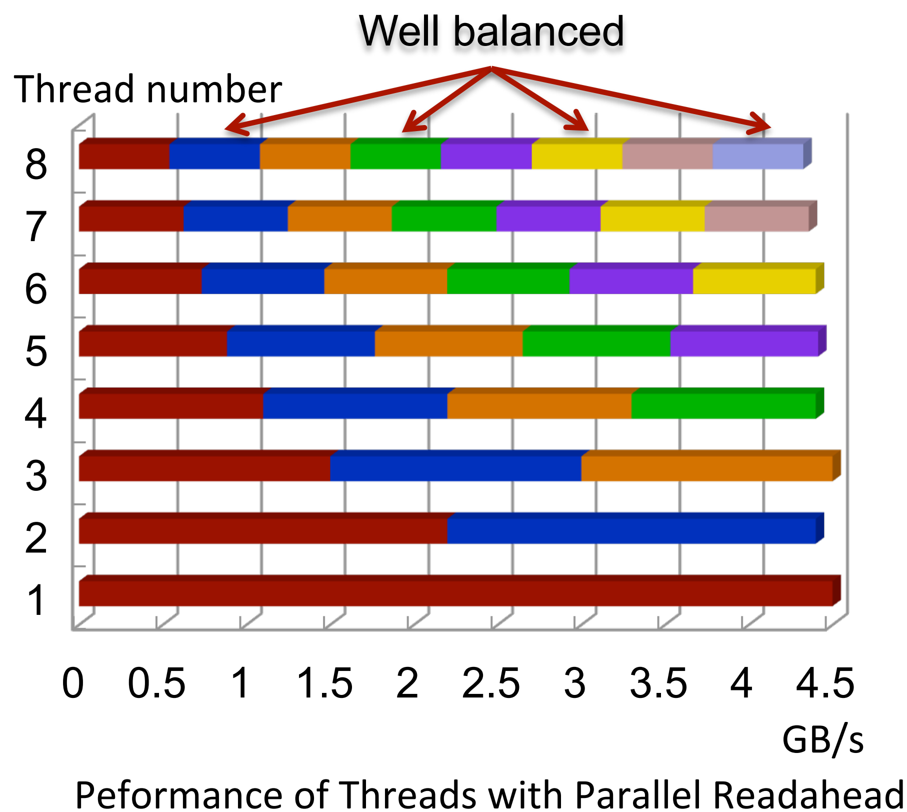
- **One thread read 50 GB file with different I/O sizes**
  - dd if=/lustre/file of=/dev/null bs=$IO_SIZE



Benchmarks: Multiple thread read

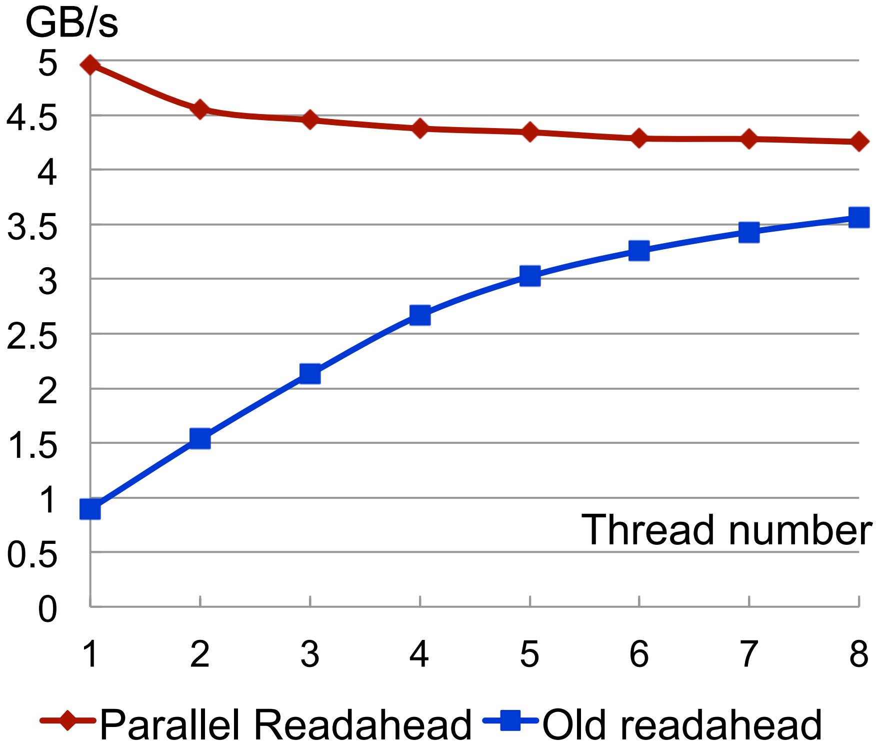
- **All threads read separate 50GB files at the same time**
  - dd if=/lustre/file_${thread_index} of=/dev/null bs=1048576 &

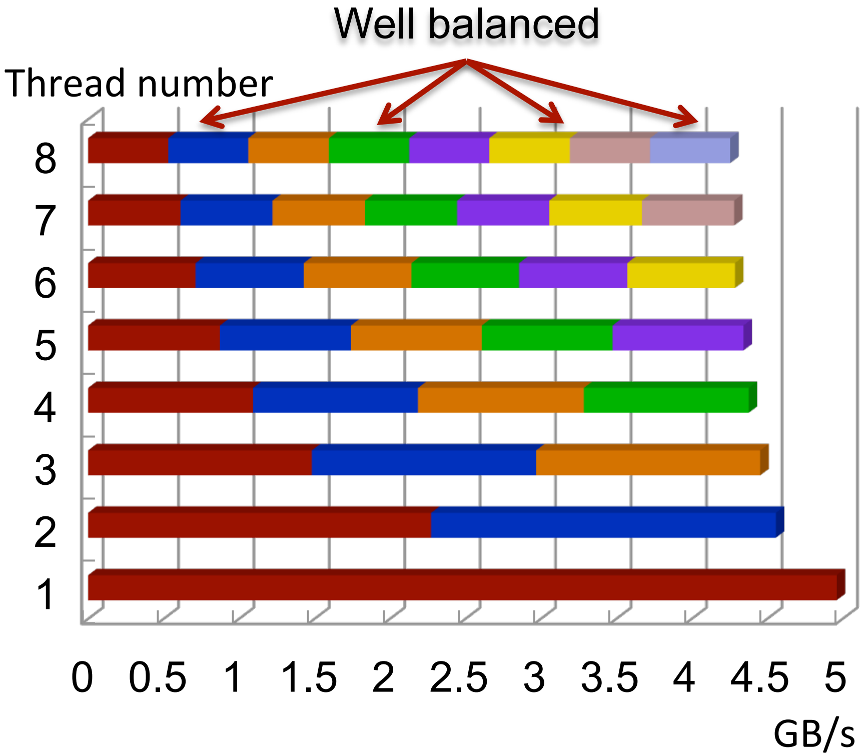




Benchmarks: Multiple thread stride read

- **All threads read separate 50GB files at the same time, read 1MB and then skip 1MB**

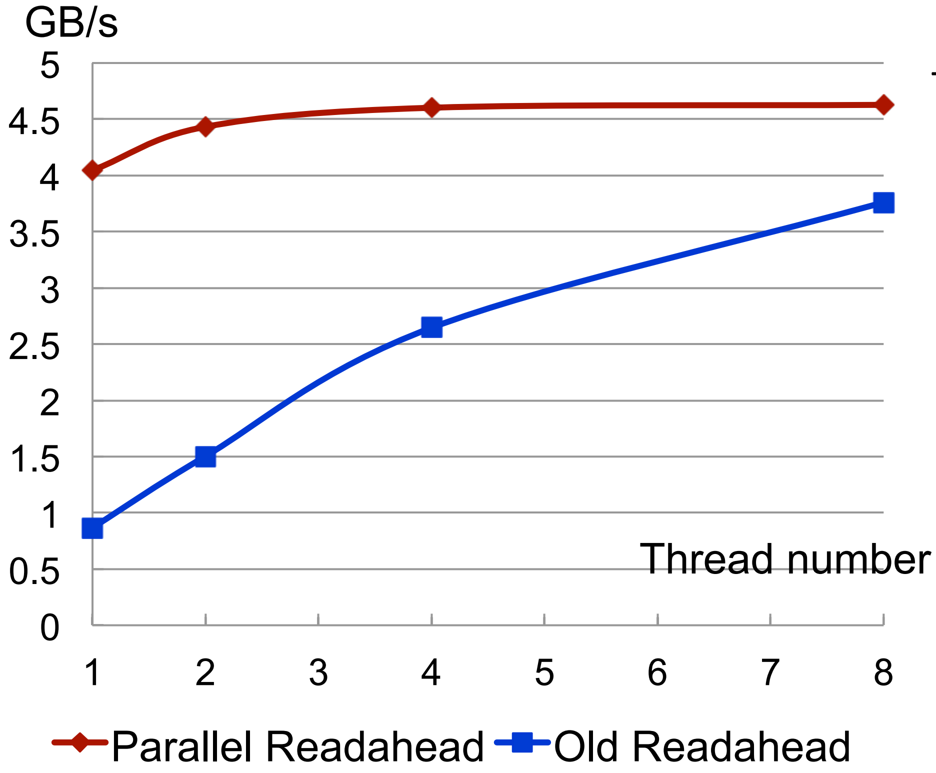


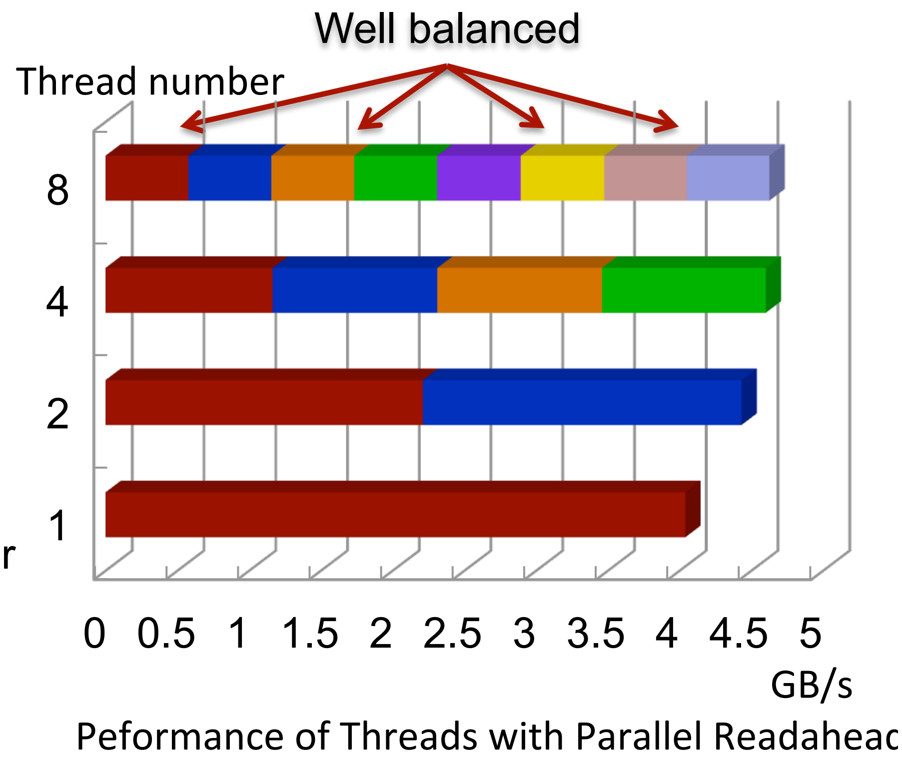


Peformance of Threads with Parallel Readahead

Benchmarks: Multiple thread stride read

- All (N) threads read separate 400/N GB files at the same time, read 1MB and then skip 6MB





Further work

- **More benchmarks**
  - Random read
  - Mixed patterns in a single thread
  - Mixed patterns in multiple threads
  - Real applications that are not only I/O intensive but also CPU and memory intensive
- **Combine pattern detection with lock ahead feature (LU-6148)**
  - To improve access performance of shared file I/O from multiple clients
- **Single thread write improvement**
  - Client side latency is the main cause of slow single thread write
  - Patches that bypasses write on OSD(LU-7655) could simplify benchmarking a lot

Thank you!