Whamcloud

Buffered I/O or Direct I/O, that is the question

Qian Yingjin
qian@ddn.com

Nov. 8rd, 2024

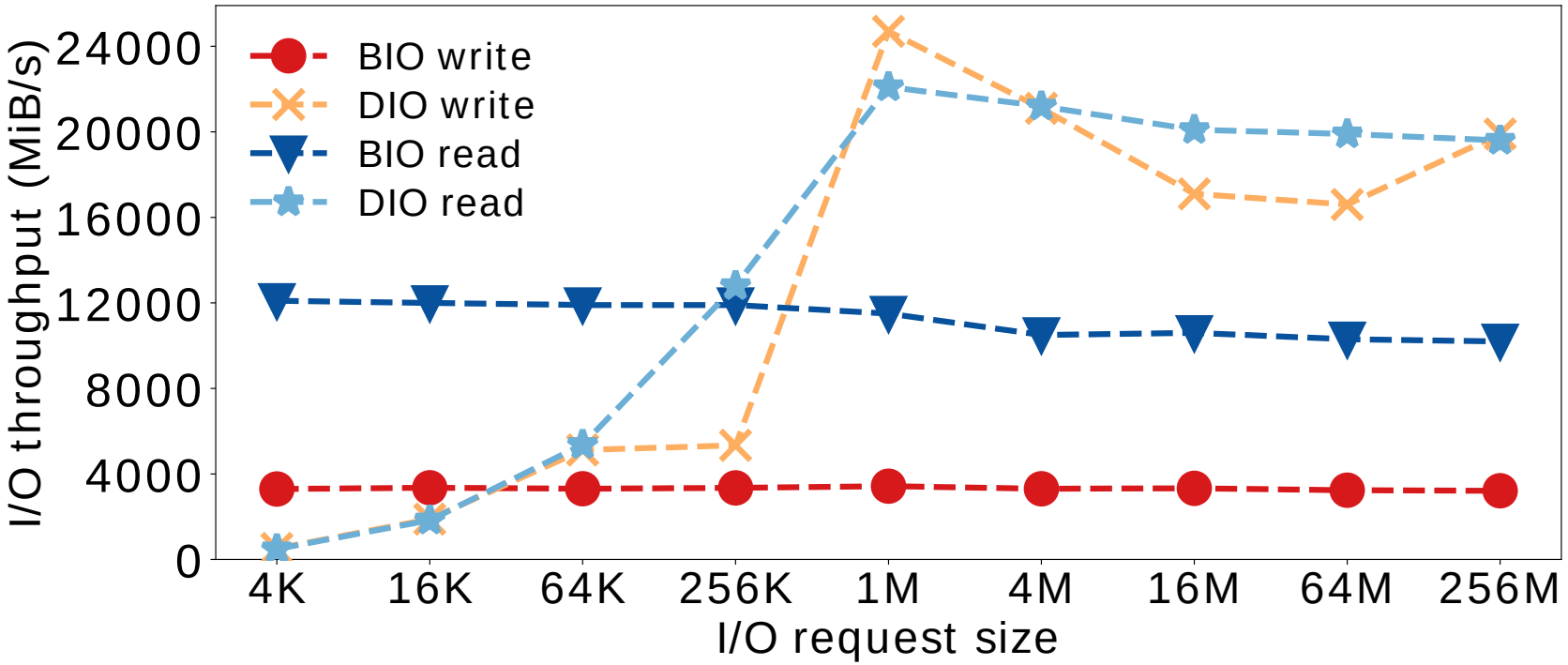
Outline

Whamcloud

- Background and motivation
- autoIO design and implementation
  - Unaligned direct I/O support
  - Client-side I/O mode decision
  - Server-side write-back
  - Cross-file batching for buffered writes
  - Delayed allocation on OSD layer (ldiskfs)
- Performance evaluation
- Conclusion and future work

Background and Motivation

- Linux's default I/O mode is *buffered I/O* utilizing the page cache
- The alternative *direct I/O* bypasses the Linux page cache and can be more beneficial

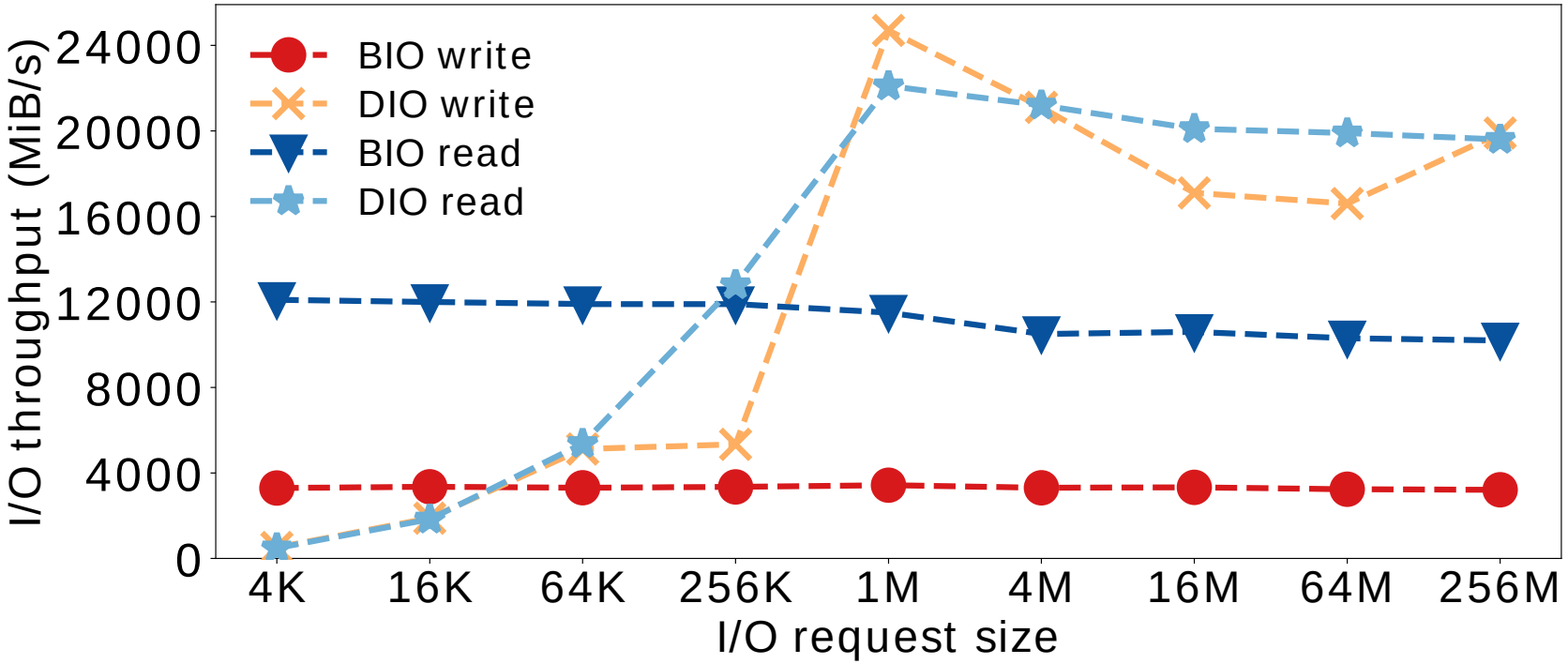


Local ldiskfs performance for various I/O sizes

Motivation

Whamcloud

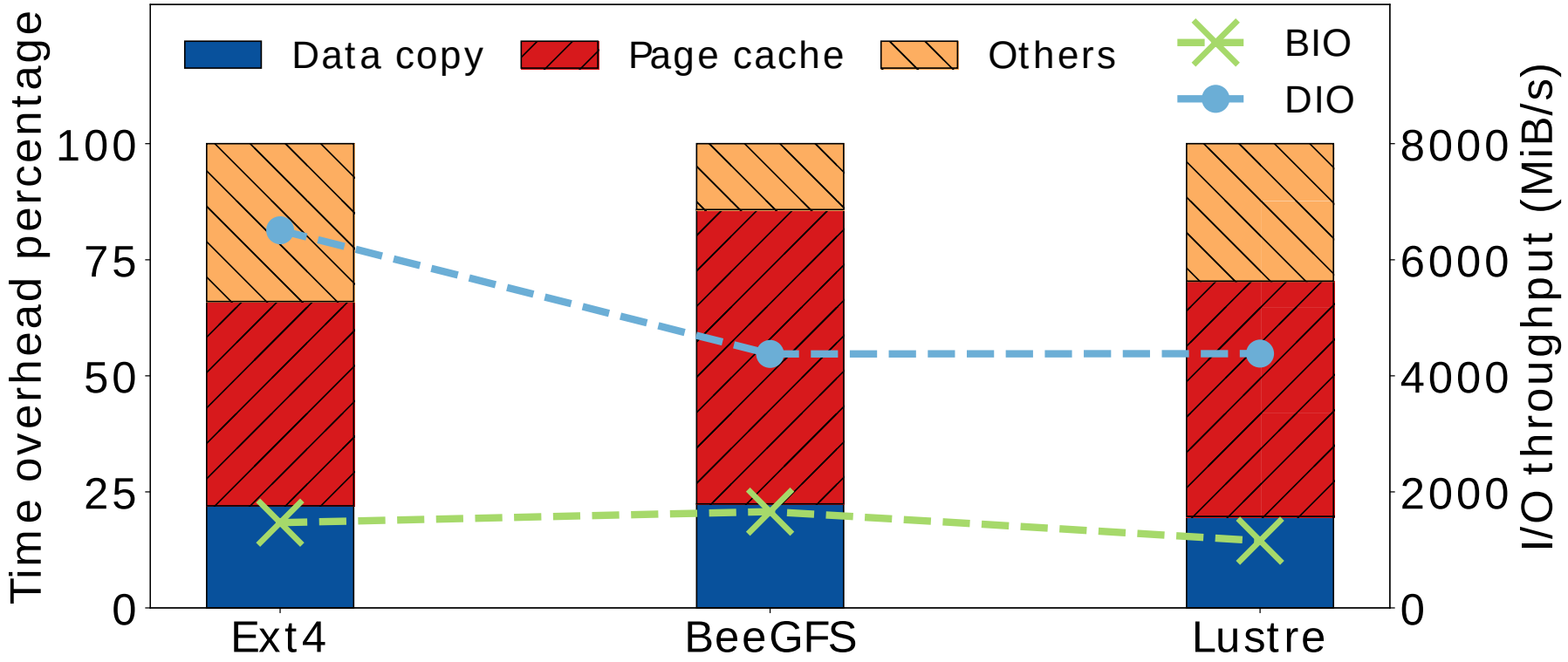
- Various challenges hinder higher direct I/O adoption
  - Users tend to use the familiar I/O mode
  - Alignment constraints can be difficult to accommodate
  - It is often unclear which I/O mode performs better



Background The cost of page caching in buffered I/O

Whamcloud

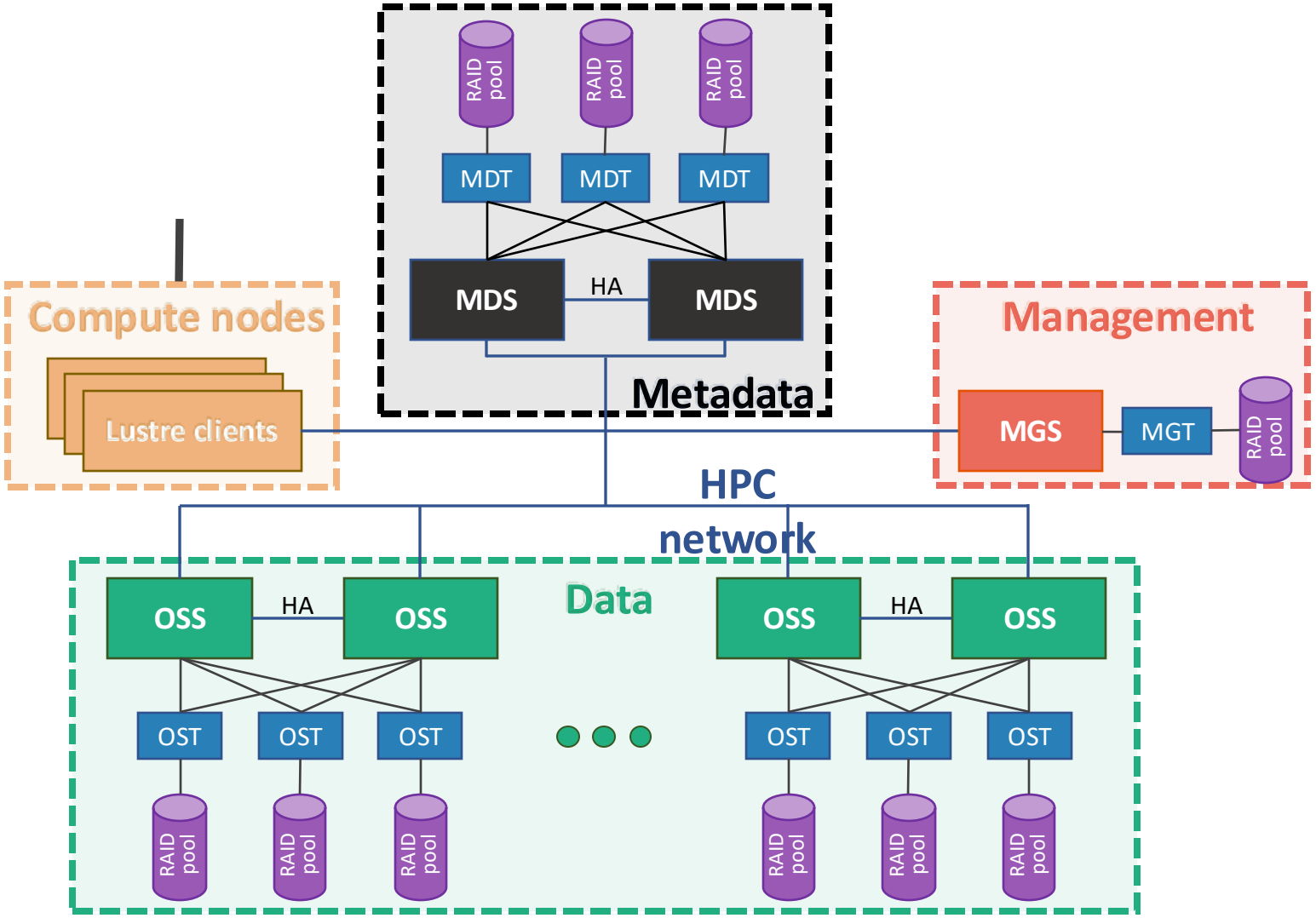
- Only about 20% of the overall time is spent copying data
- More than 40% of the overall time is spent on page cache management



I/O time breakdown for buffered I/O writes (16 MiB I/O size) via perf

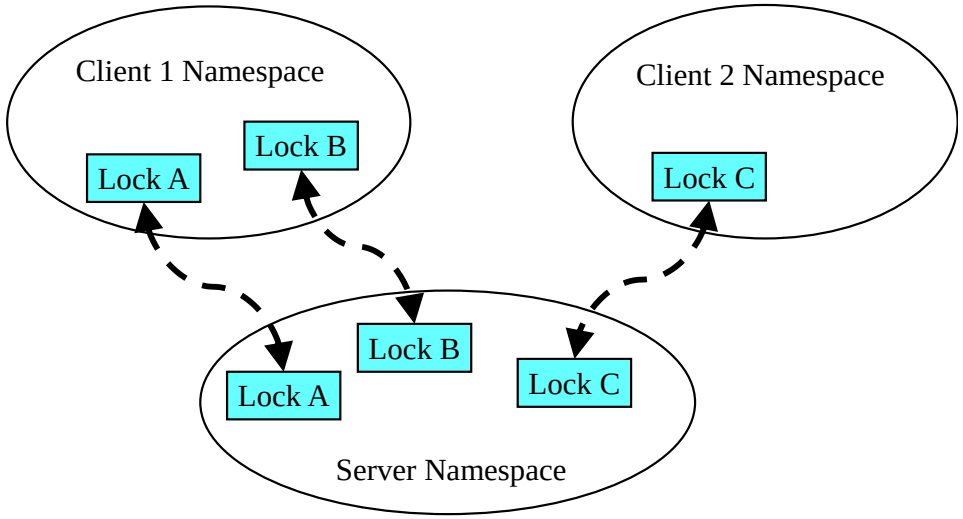
Background Lustre basic architecture

- MDS: Metadata Server
  - Processes metadata requests
- MDT: Metadata Target
  - Stores metadata content
  - Link, permission, attributes
- OSS: Object Storage Server
  - Processes I/O requests
- OST: Object Storage Target
  - Stores data content
  - File sizes, block count, mtime



Background Lustre Distributed Lock Manager (LDLM)

- LDLM is used for file synchronization and metadata access
- A lock corresponds to a certain resource ID in a namespace (NS)
  - Each Lustre target (OST, MDT, MGT) has a DLM namespace
  - Each Lustre target has full authority about its namespace
- Clients have a copy of a server lock for locally-accessed resource (shadow namespace)

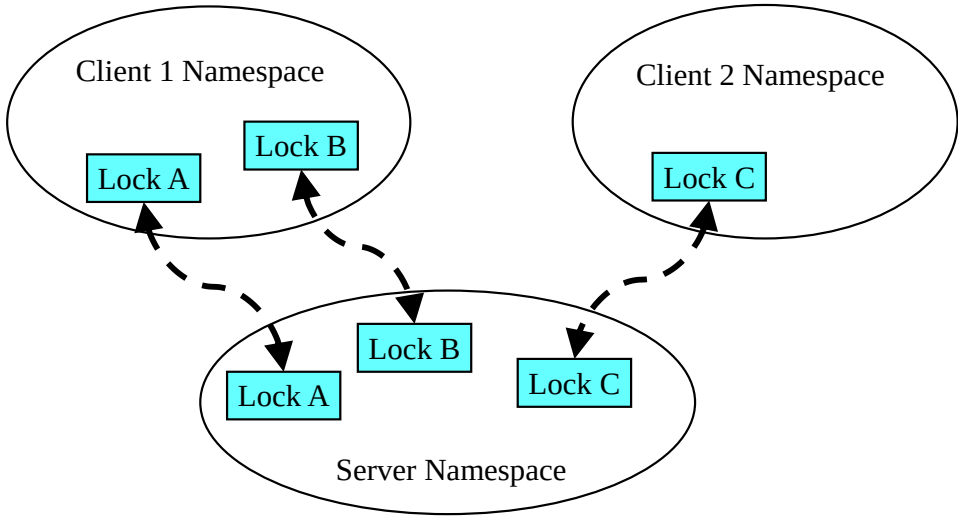


Shadow namespaces on clients

Wang et al. "Understanding Lustre filesystem internals.", 2009

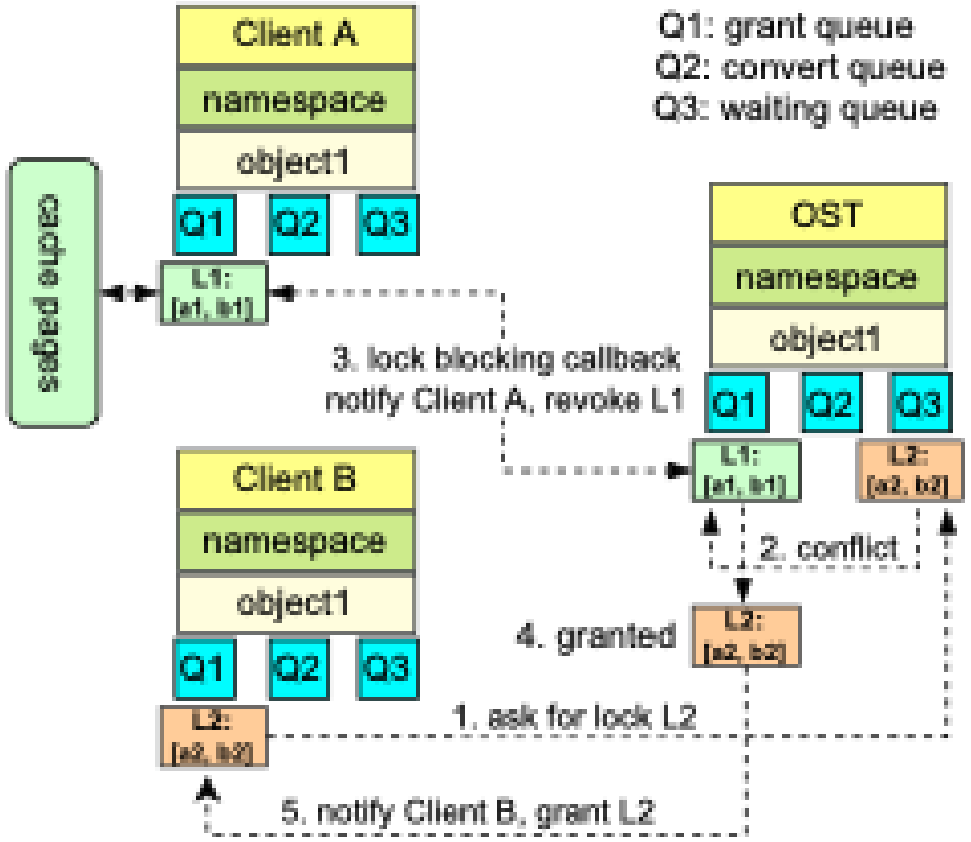
Background Lustre Distributed Lock Manager (LDLM)

- LDLM is used for file synchronization and metadata access
- A lock corresponds to a certain resource ID in a namespace (NS)
  - Each Lustre target (OST, MDT, MGT) has a DLM namespace
  - Each Lustre target has full authority about its namespace
- Clients have a copy of a server lock for locally-accessed resource (shadow namespace)



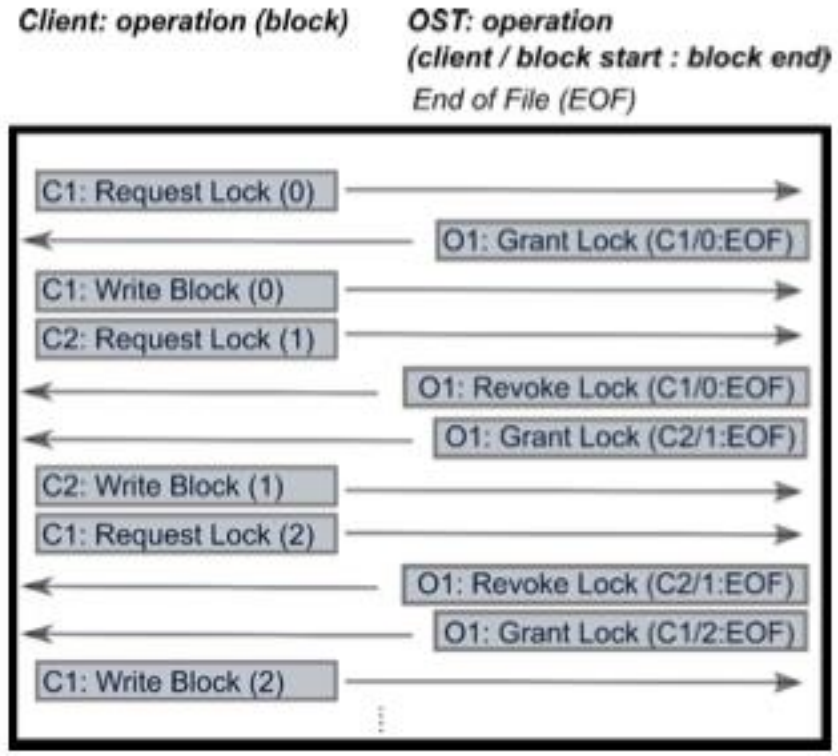
Shadow namespaces on clients

Wang et al. "Understanding Lustre filesystem internals.", 2009



Lock callback overhead under conflict access

Background - Lustre I/O Locking Overview

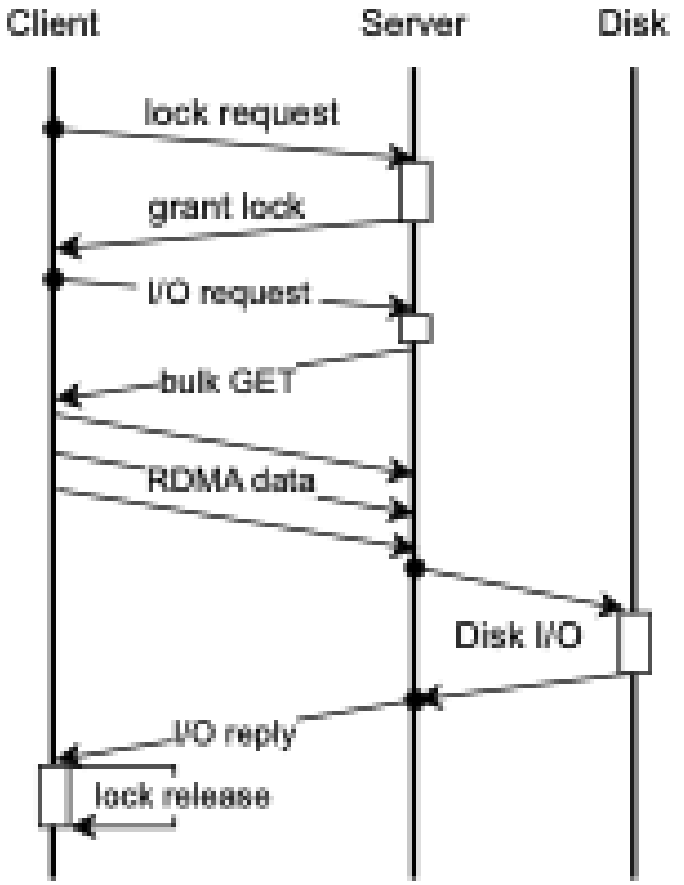


DLM Locking under IOR SSF I/O pattern

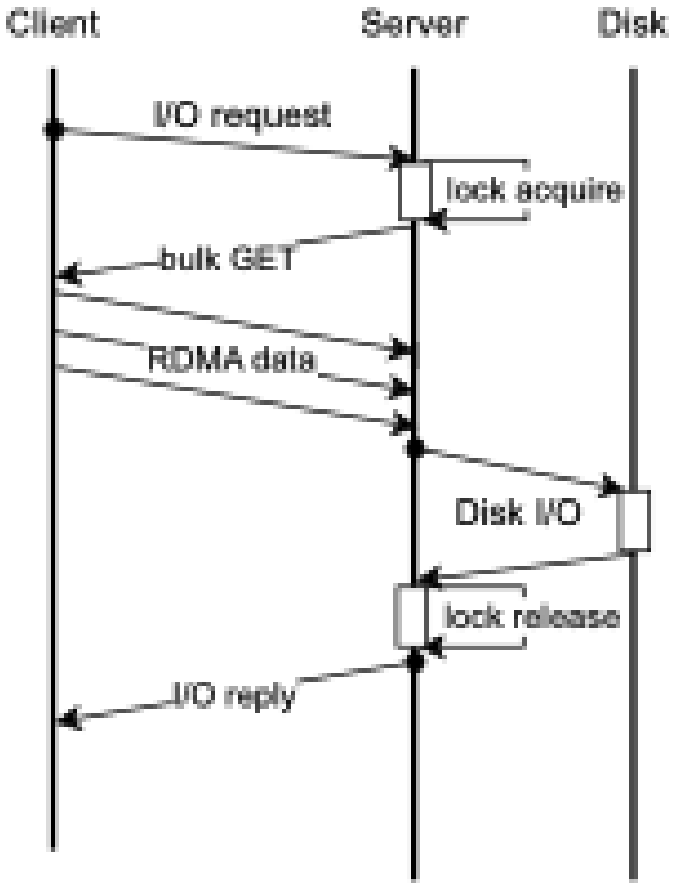
Michael Moore, Patrick *et al. "Lustre Lockahead:* Early Experience and Performance using Optimized Locking *."*

- Multiple Lustre clients accessing a shared file requires locking between clients.
- Multiple clients accessing the same OST object cause lock contention.
- Extent Locking policy:
  - Default Lustre locking expands locking extent.
  - Lock expansion leads to lock contention due to false sharing.
  - However, lock expansion does improve performance over no lock expansion.
  - Lock-ahead can only solve the lock conflict in stride I/O mode.

Background – Server Side Locking for direct I/O



Client/server locking for buffered I/O



Server-side locking for direct I/O

- Take extent lock on server side;
- Shorter locking path;
- No lock extent expansion;
- Avoid heavy lock callback for concurrent conflict shared access;

Insight: Server-side locking for direct I/O is much efficient

Weighing I/O modes in Lustre

Whamcloud

I/O case	Buffered I/O	Direct I/O
Small I/O size	✓	✗
High latency storage	✓	✗
Unaligned I/O	✓	✗
Large sequential I/O	✗	✓
Many running processes/nodes	✗	✓
System under memory pressure	✗	✓

Weighing I/O modes in Lustre

Whamcloud

I/O case	Buffered I/O	Direct I/O
Small I/O size	✓	✗
High latency storage	✓	✗
Unaligned I/O	✓	✗
Large sequential I/O	✗	✓
Many running processes/nodes	✗	✓
System under memory pressure	✗	✓

Feature	Abbreviation
Original Lustre	vanilla
Unaligned direct I/O support	-
Client-side I/O mode decision	autoIO
Server-side write-back	svrWB
Cross-file batching for buffered writes	XBatch
Delayed allocation	delalloc

- Areas of improvement:
  - Use best of both worlds in a given situation (only switch from BIO -> DIO)
  - Remove direct I/O alignment constraints
  - Improve many small file performance and reduce file fragmentation

Unaligned DIO: Buffer, no cache

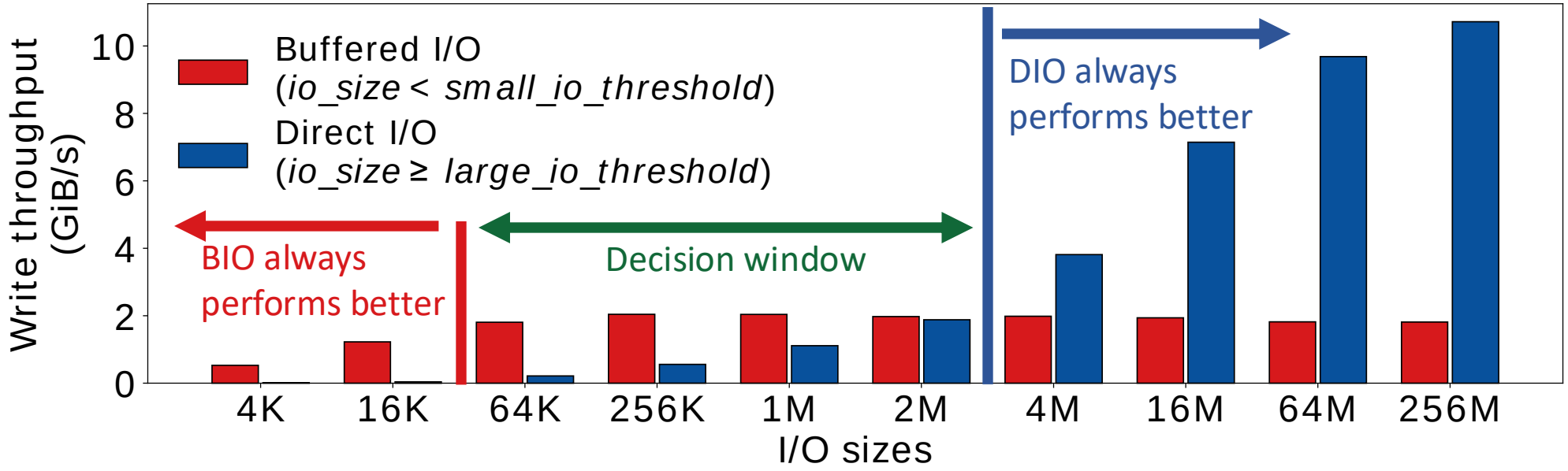
Whamcloud

- **Buffered I/O:**
  - Page cache is naturally aligned, but is very expensive
  - Copy unaligned data into aligned pages;
  - A cache can be used repeatedly and accessed from multiple threads;
    - Require lots of concurrency management and locking
    - Most cost of cache is not in data copying – cost is in cache setup
- **Unaligned Direct I/O:**
  - To get alignment:
    - Allocate an aligned buffer;
    - Copy data to/from the buffer;
    - Do direct I/O form the buffer;
  - I/O is still synchronous – when write()/read() returns , I/O is completed, and buffer is released;
  - Buffer is not accessible from other thread;
  - No need for cache setup or locking

AutoIO

- Automatic alignment of unaligned direct I/O
- Primary I/O mode decision based on I/O size
- Two I/O thresholds allow a *decision window* for
  - Lock contention
  - Memory pressure and low cache reusage
  - Default decision window: [32 KiB, 2 MiB)

Feature	Abbreviation
Unaligned direct I/O support	-
Client-side I/O mode decision	autoIO



AutoIO algorithm

Whamcloud

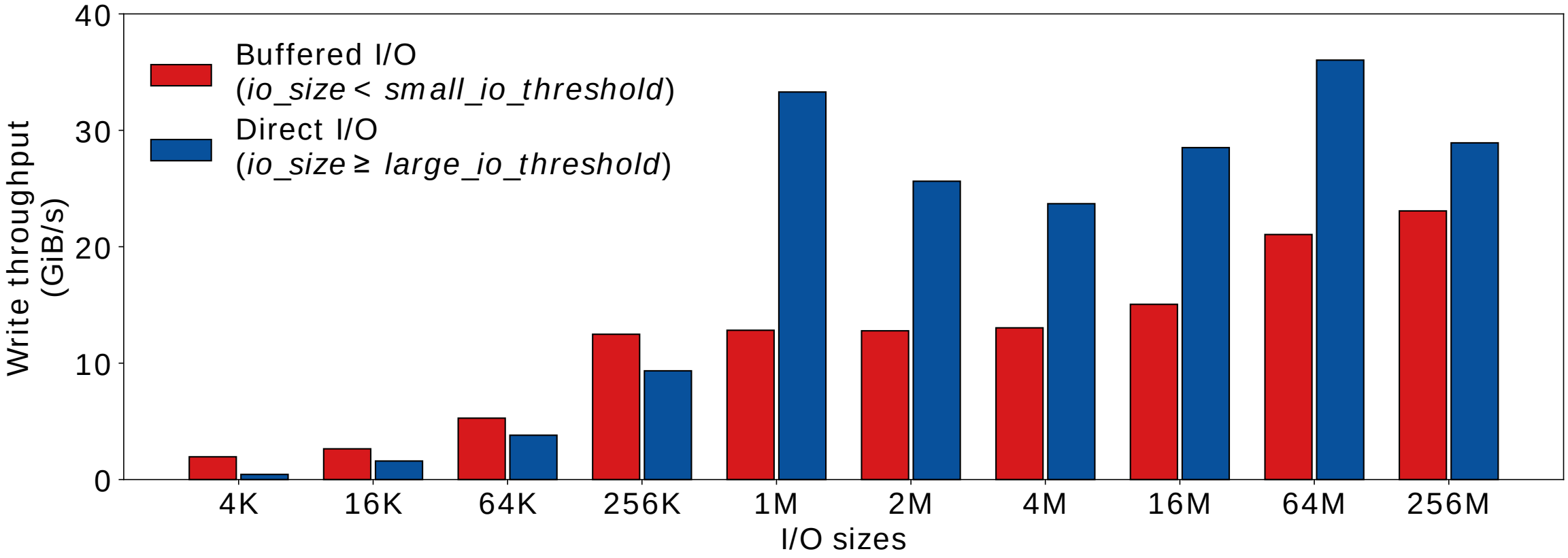
Algorithm 1 The autoIO decision algorithm

```
1:  function DECIDE_IO_MODE(file, io_size, cfg)
2:      if (io_size ≥ cfg.large_io_threshold) then
3:          return DIO;
4:      else if (io_size < cfg.small_io_threshold) then
5:          return BIO;
6:      else if file_is_under_lock_contention(file) then
7:          return DIO;
8:      else if client_is_under_memory_pressure(file, cfg) then
9:          return DIO;
10:     else if file_lacks_access_locality(file, io_size, cfg) then
11:         return DIO;
12:     else
13:         return BIO;
```

I/O modes under lock contention

Whamcloud

- Lock contention workload: Strided I/O over 10 nodes
- Under extreme lock contention, direct I/O becomes beneficial earlier

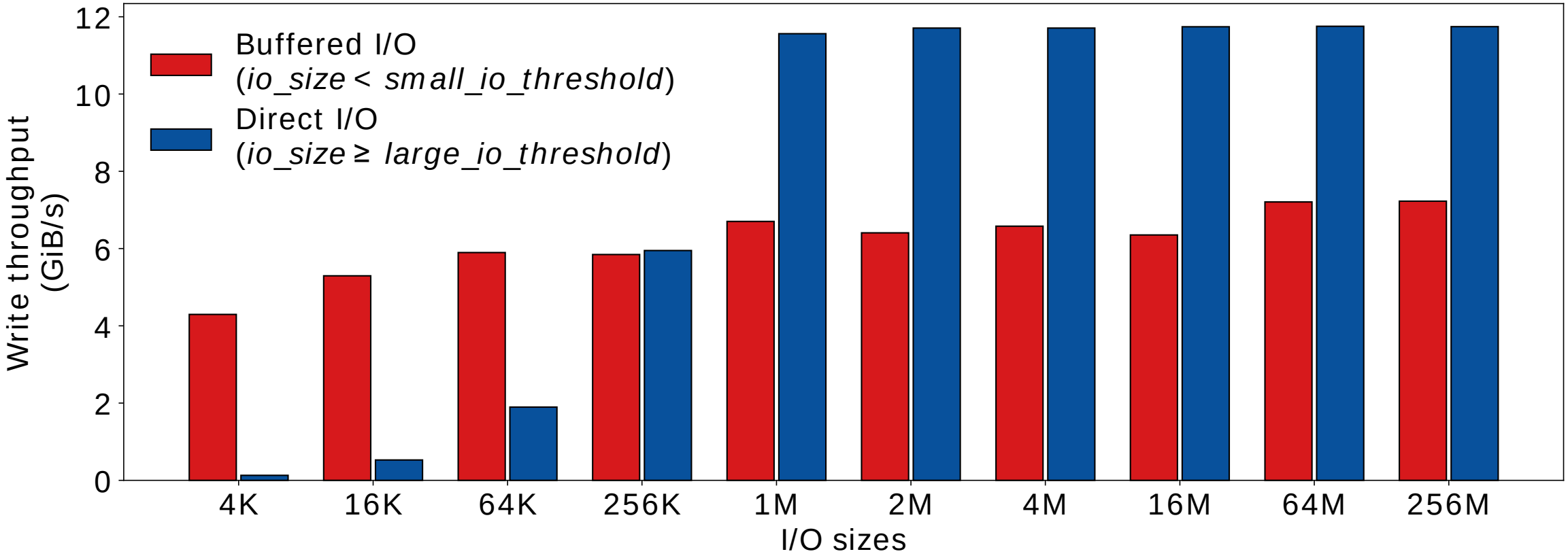


I/O throughput for various I/O sizes and I/O modes under lock contention

I/O modes under cache overusage

Whamcloud

- Over caching workload: cached pages are not reused, memory is under pressure
- Under memory restrictions, direct I/O becomes beneficial earlier



I/O throughput for various I/O sizes and I/O modes under cache over usage

Further optimizations

Whamcloud

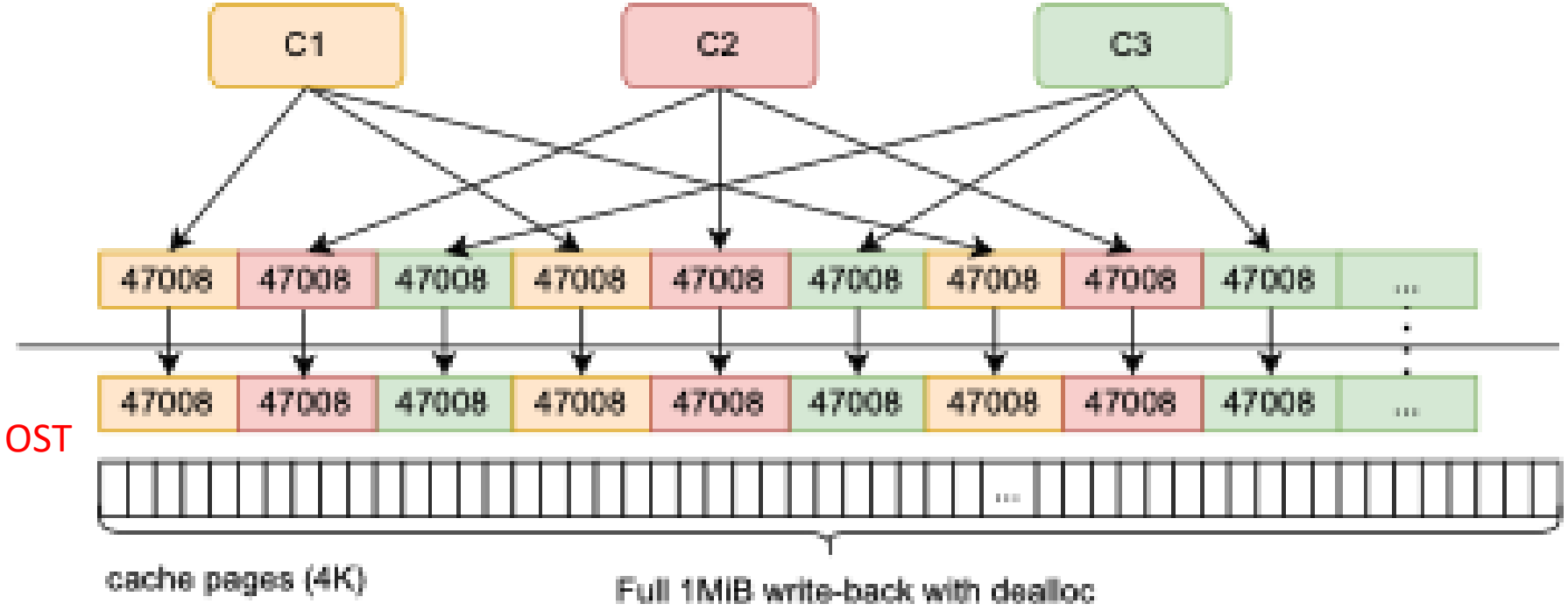
- Server-side write-back
  - Vanilla Lustre uses write-through
  - Lustre servers can switch to write-back at a threshold
- Cross-file batching for buffered writes
  - Vanilla batches dirty pages into large bulk RPCs (1MiB)
  - This can improve network and disk efficiency
  - But, it can prolong flush operations of many small files
  - Batch dirty pages of multiple files into one large bulk RPC

Feature	Abbreviation
Server-side write-back	svrWB
Cross-file batching for buffered writes	XBatch
Delayed allocation	delalloc

Further optimizations

- Delayed allocation (osd-ldiskfs)
  - Improves svrWB further to reduce file fragmentation
  - File fragmentation can be caused during strided writes to a single file from many clients
  - Delayed allocation collects and merges small and non-contiguous I/O request into large I/O request

Feature	Abbreviation
Server-side write-back	svrWB
Cross-file batching for buffered writes	XBatch
Delayed allocation	delalloc



Evaluation

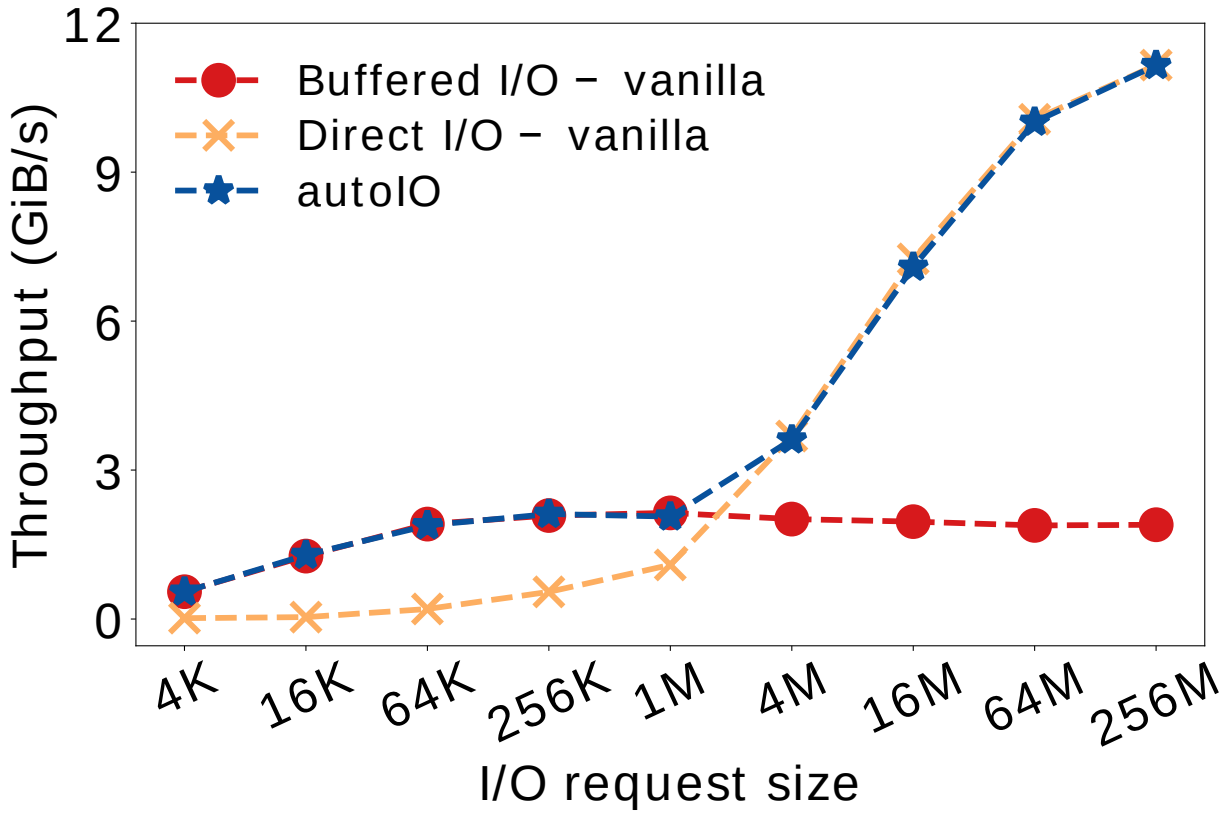
Whamcloud

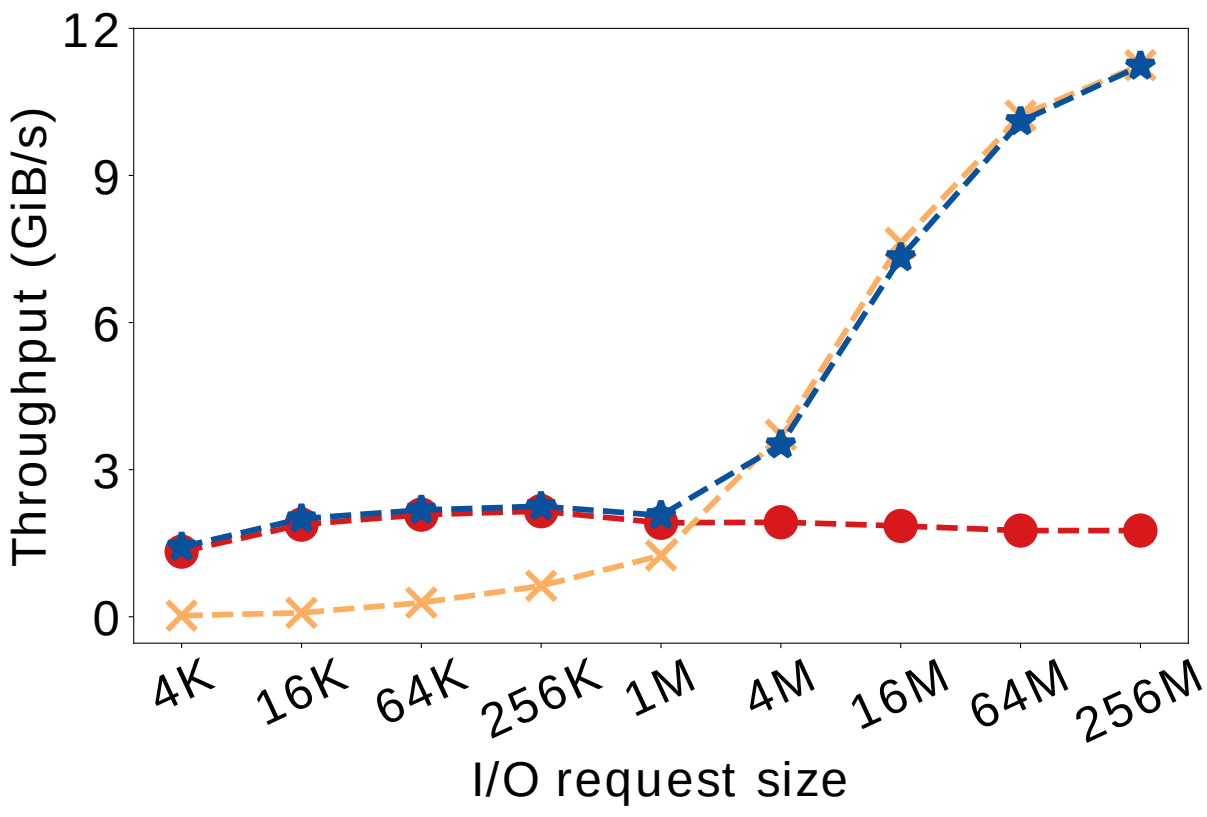
- Lustre 2.15.58 cluster with CentOS 8.7
  - 4 Meta Data Target (MDT)
  - 8 Object Storage Targets (OSTs)
  - Servers using DDN AI400X Appliance (20x Samsung 3.84 TiB NVMe, 4×IB-HDR100)
  - 32 client nodes using Intel Gold 5218 processor, 96 GiB DDR4 RAM, IB-HDR 100, 1 Gbps Ethernet
- BeeGFS 7.4.0
  - Offers two client-side file cache modes
    1. **buffered** (default): Write-back and read-ahead using static buffers
    2. **native**: Relies on the Linux page cache - switches to direct I/O on a set I/O threshold (512 KiB)
- OrangeFS 2.10.0
  - Offers two server-side I/O modes
    1. **alt-aio** (default): Buffered asynchronous I/O
    2. **directio**: Direct I/O mode

Single process I/O streaming

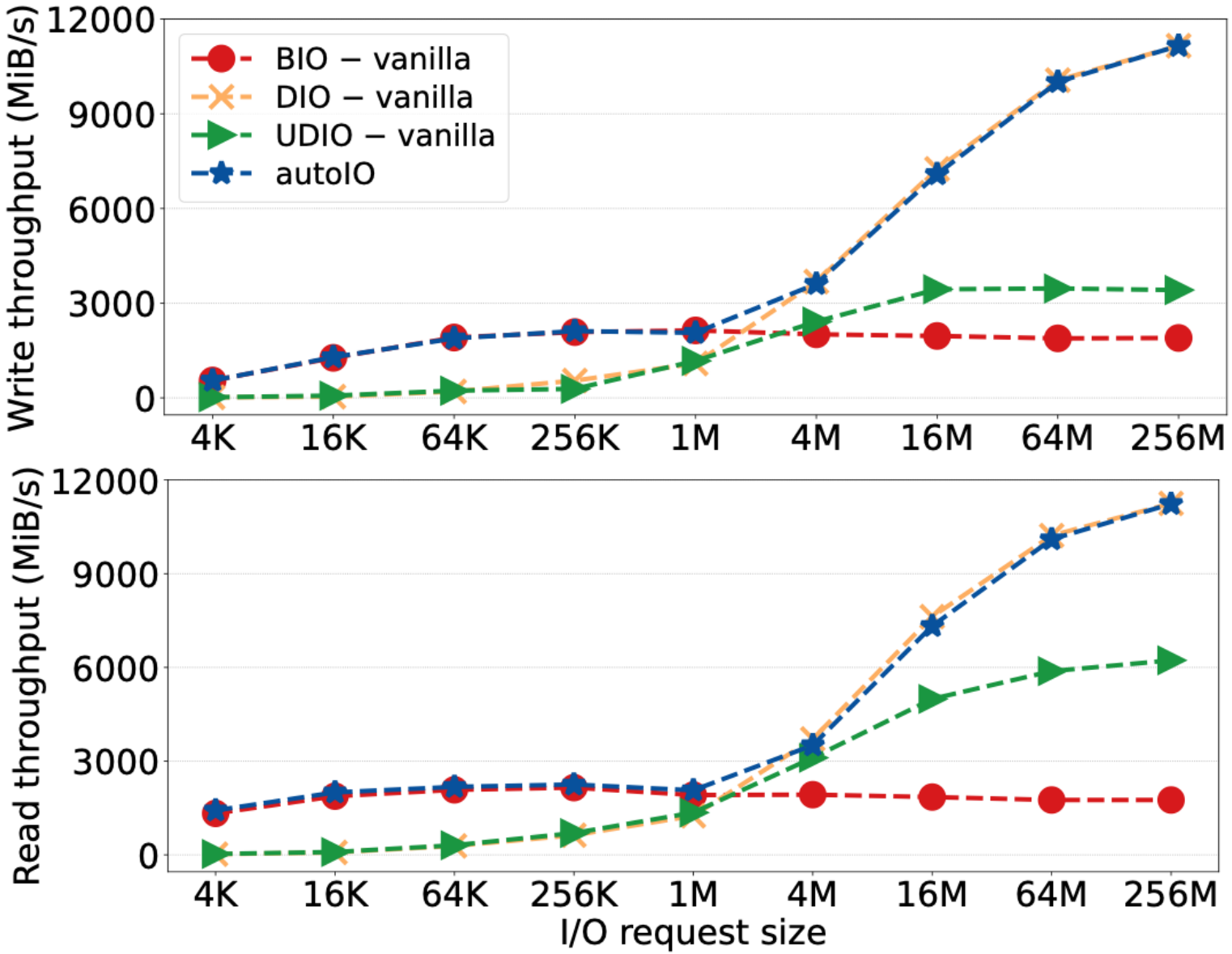
Whamcloud

- Represents the main use case of autoIO: sequential I/O for a single process





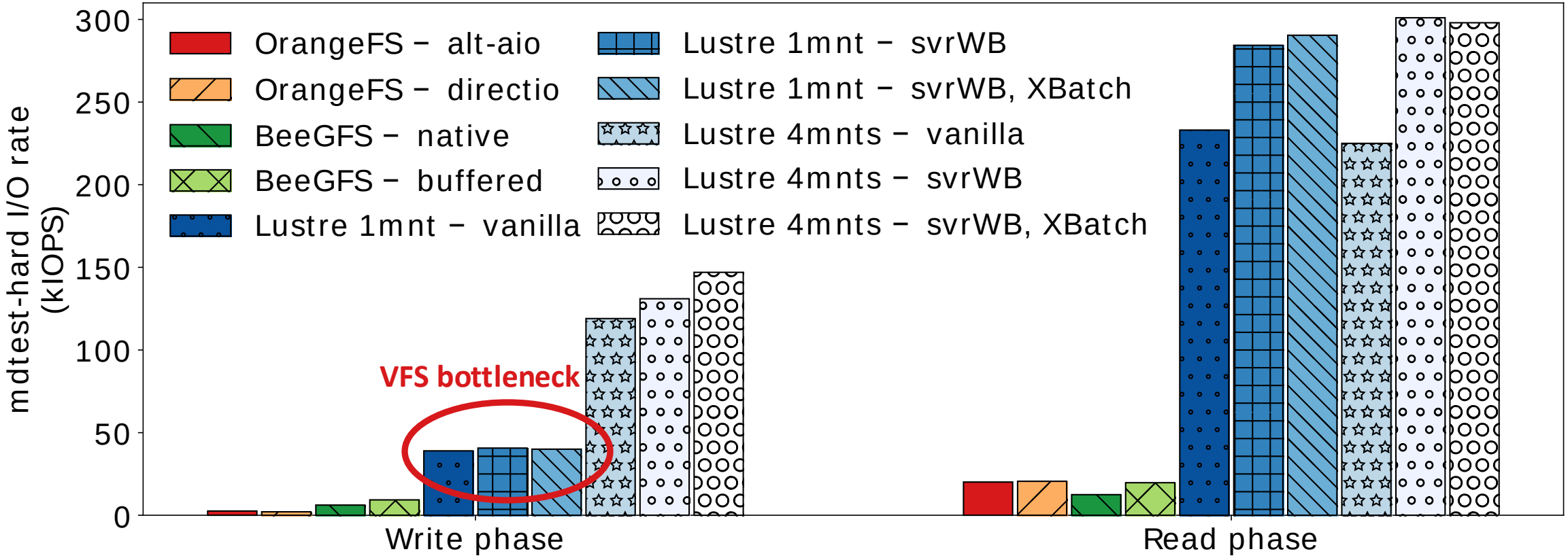
Unaligned direct I/O (UDIO)



IO500 (10-node challenge: mdtest-hard)

Whamcloud

- mdtest-hard generates many small files (3091 bytes in size) in a single directory
- No impact of client-side autoIO: All I/O is buffered

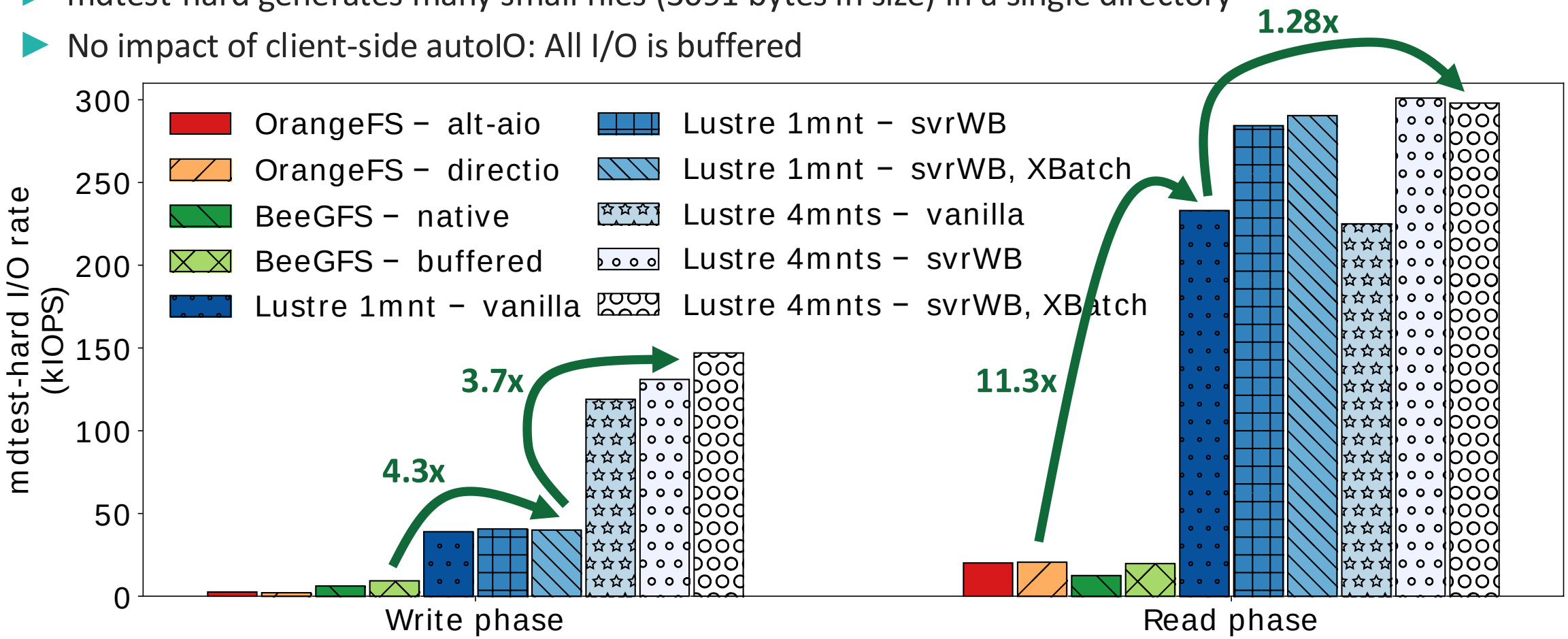


Workload for 10 clients (16 proc each) across file systems and configurations

IO500 (10-node challenge: mdtest-hard)

Whamcloud

- mdtest-hard generates many small files (3091 bytes in size) in a single directory
- No impact of client-side autoIO: All I/O is buffered

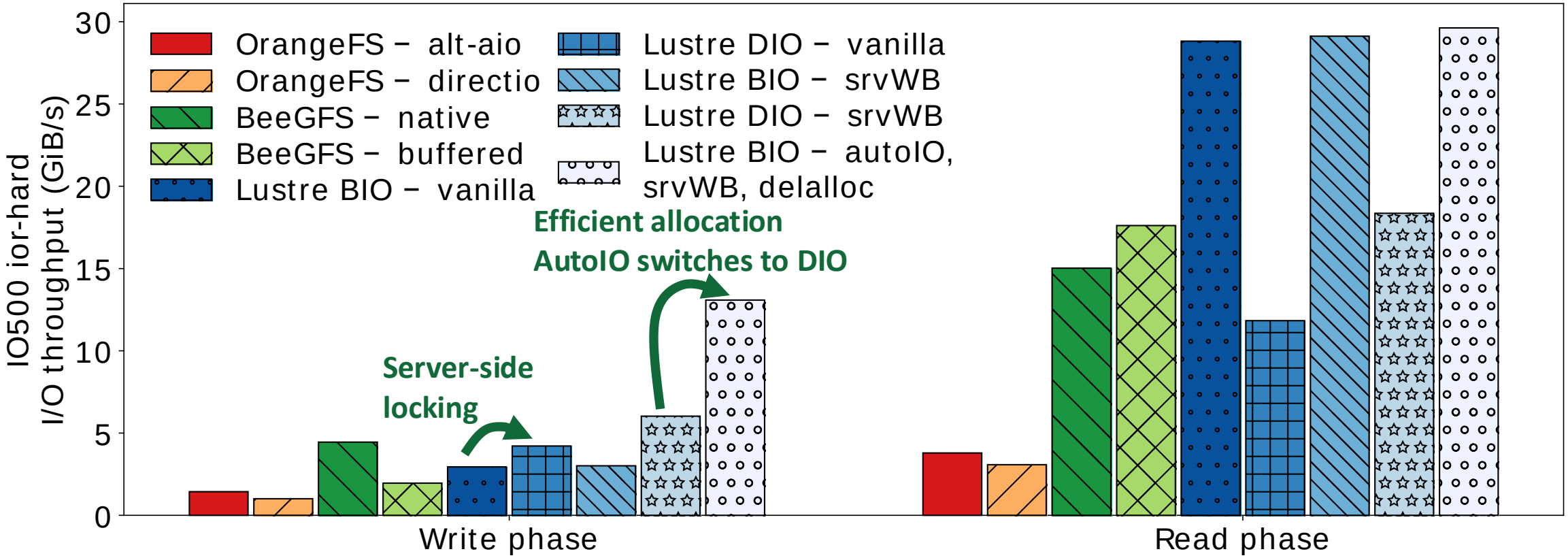


Workload for 10 clients (16 proc each) across file systems and configurations

IO500 (10-node challenge: ior-hard)

Whamcloud

- ior-hard generates unaligned, strided I/O (47,008 bytes in size) to a single shared file
- BeeGFS and OrangeFS don't support unaligned DIO => fallback to BIO

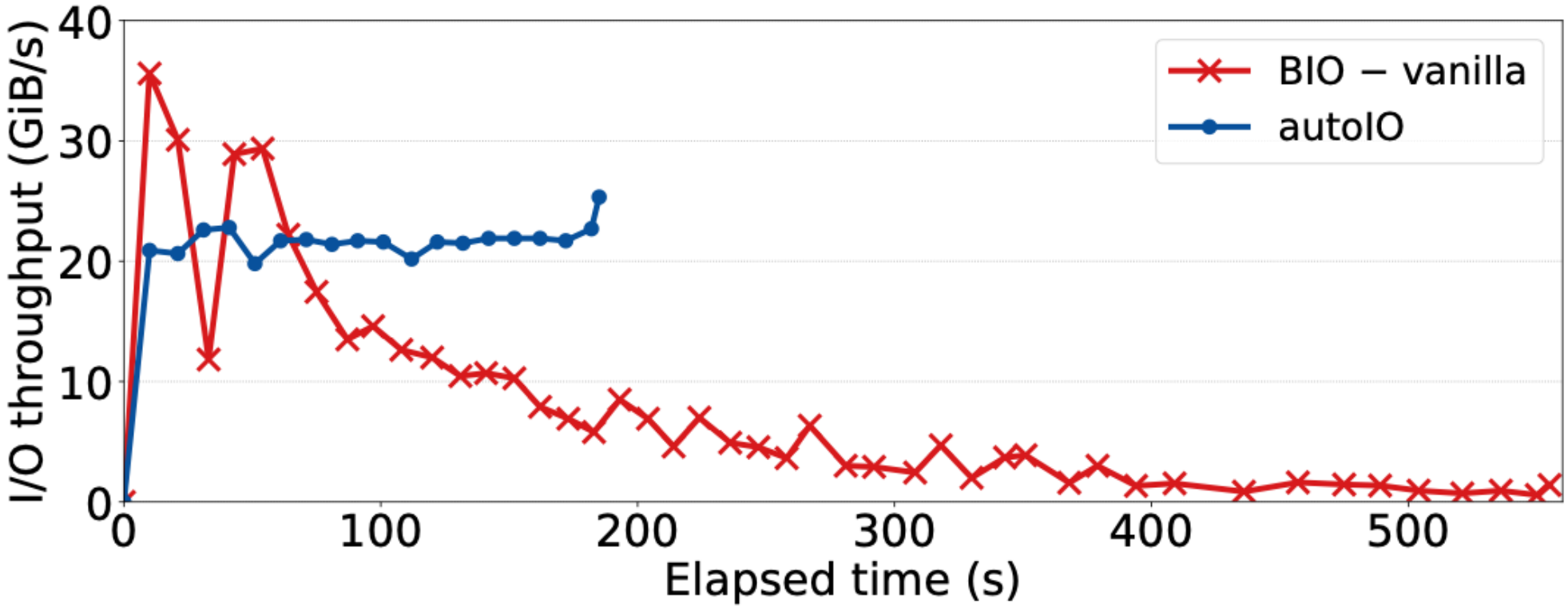


Workload for 10 clients (16 processes each) across file systems and configurations

mpiFileUtils/dcp 4TiB data

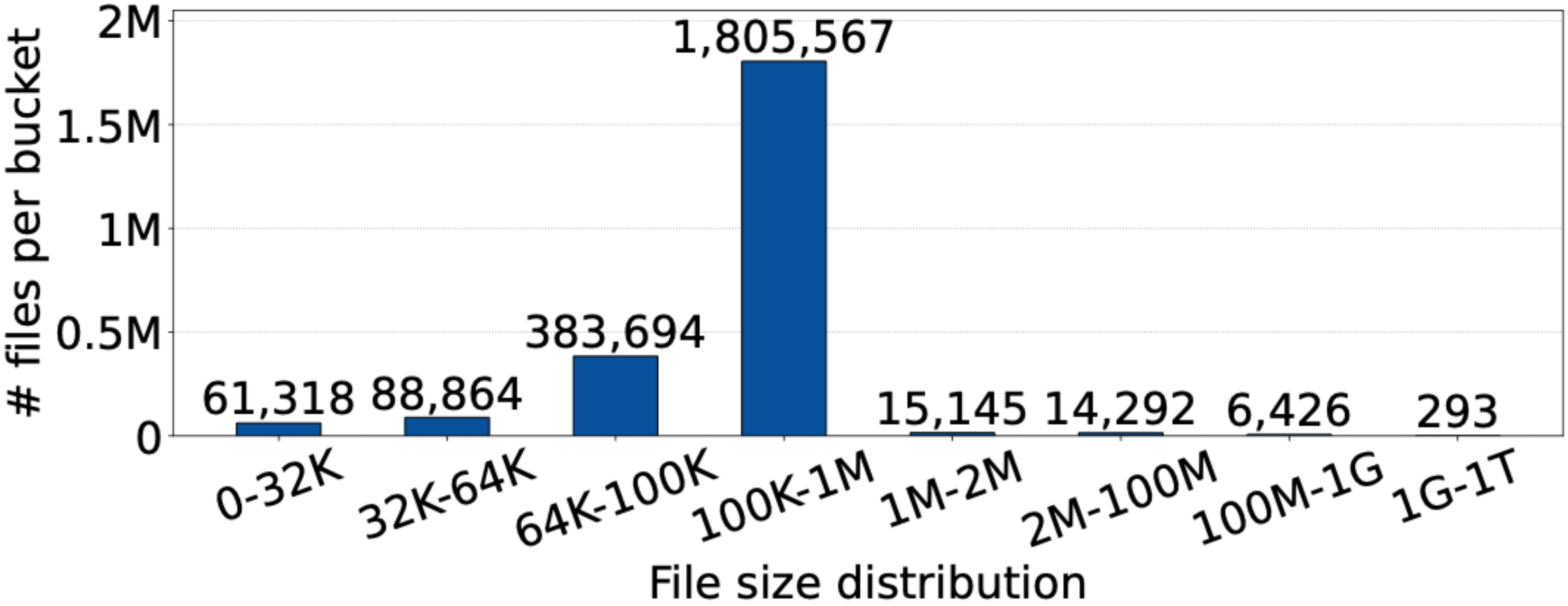
Whamcloud

- Run dcp on 32 nodes (16 processes each), writing a 4 TiB to a single shared file (same file system)
- Chunk size of 4MiB, data of the source and target files were striped over 8 OSTs



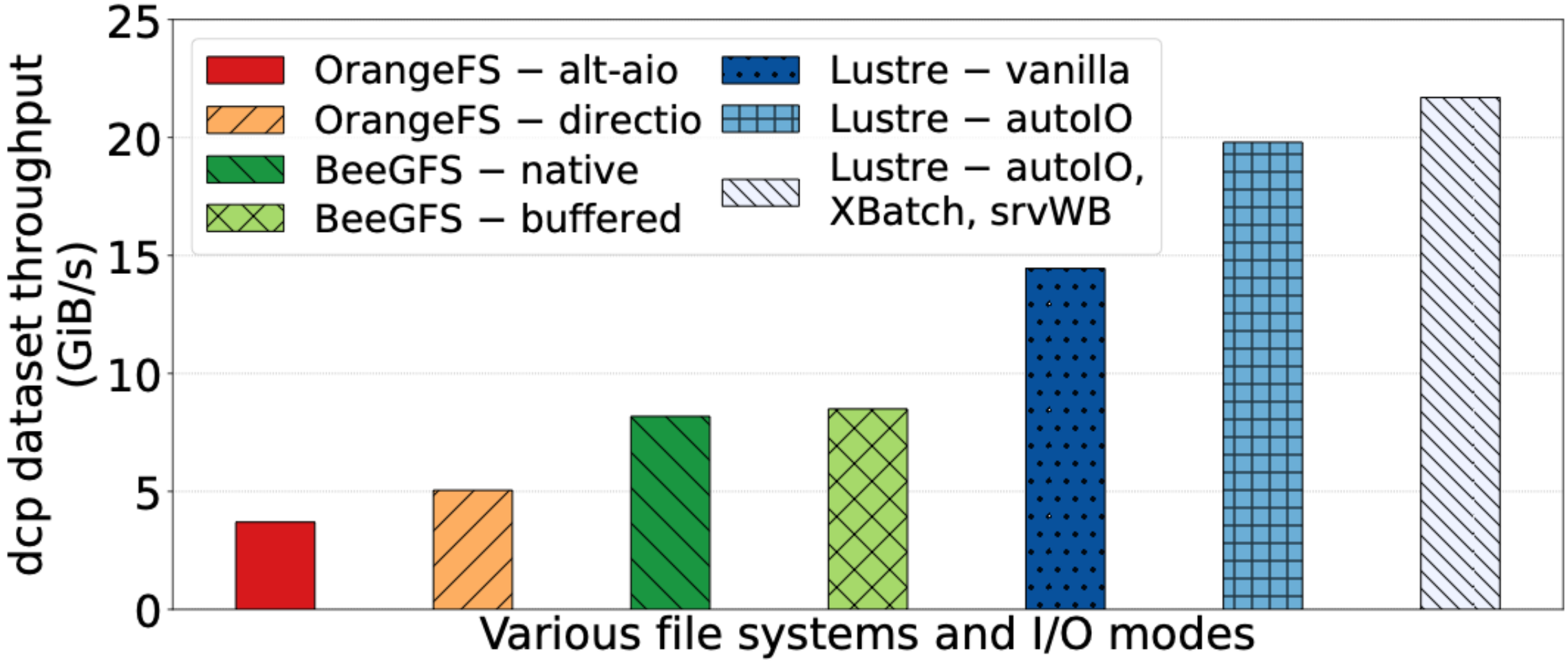
mpiFileUtils/dcp a large dataset

Whamcloud



mpiFileUtils/dcp a large dataset

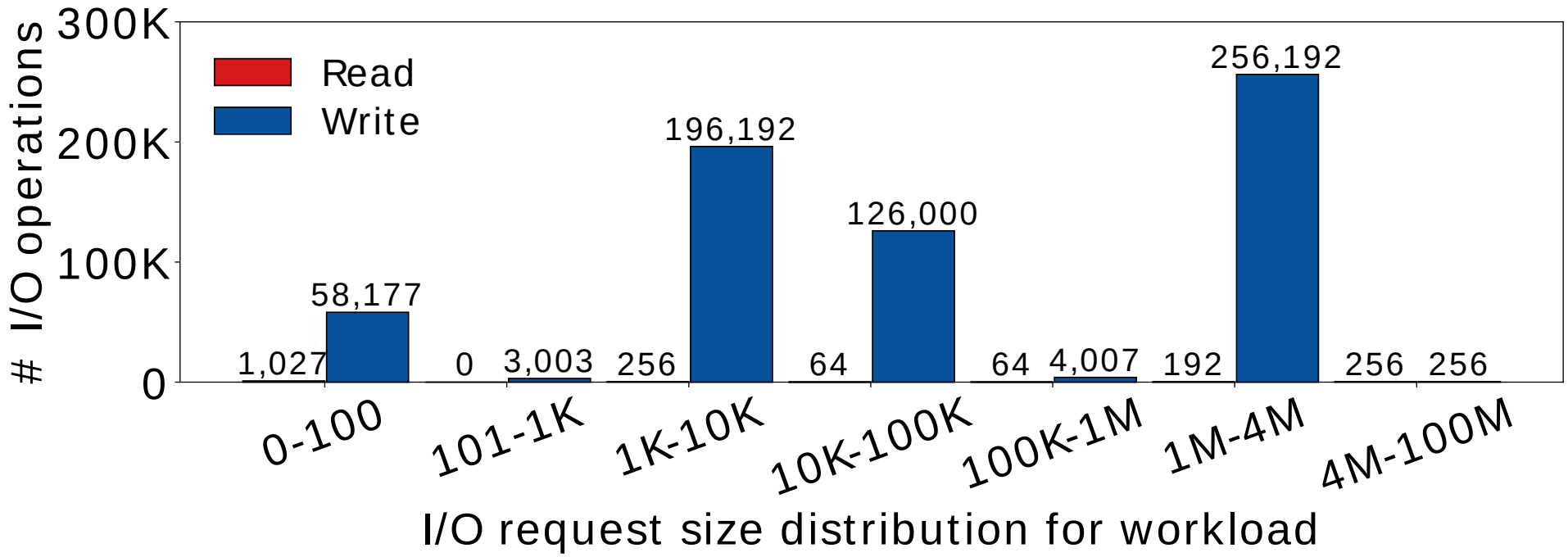
Whamcloud



Nek5000

Whamcloud

- Running the turbulent flow workload with the Nek5000 bulk-synchronous application
- 512 processes (over 32 nodes), each writing one 600 MiB file per step boundary
- 10 minute workload and a wide I/O size distribution => 600 GiB of data

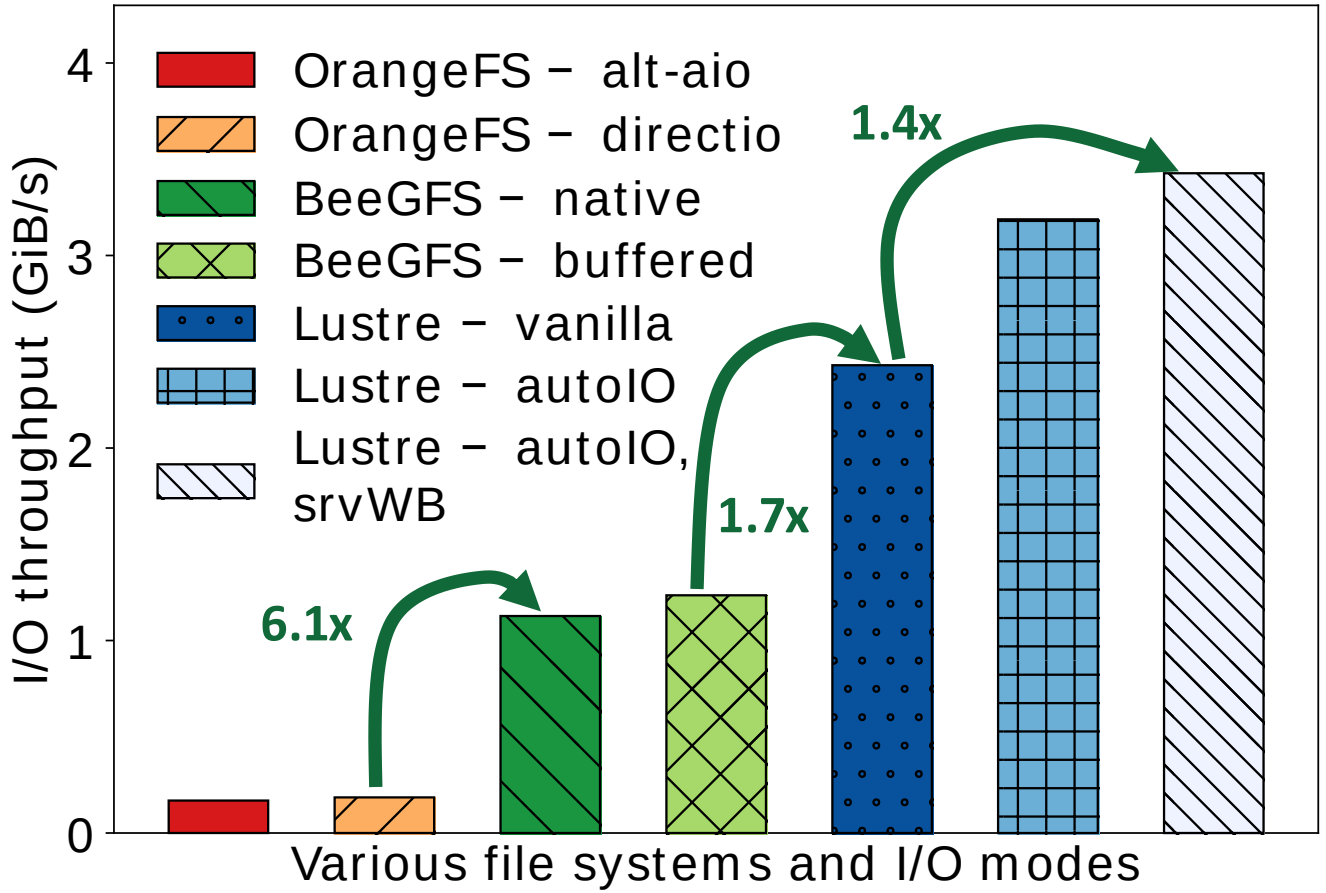


Nek5000's turbPipe workload I/O access size distribution via Darshan

Nek5000

Whamcloud

- Nek5000 turbPipe workload for 32 nodes (16 processes each)



Nek5000's turbPipe I/O throughput

I/O statistics for autoIO	Count
Buffered I/O - small threshold	327,281
Direct I/O - large threshold	128,000
Direct I/O - lock contention	132,000
Buffered I/O – default, other	65

Conclusion & Future Work

Whamcloud

- We have presented a transparent approach to combine buffered I/O and direct I/O
- We integrated our approach into Lustre keeping its strong consistency guarantees
- Key technologies: autoIO, server-side write-back, cross-file batching, and delayed alloc.
- Productization is in progress
  - Unaligned direct I/O support merged in Lustre 2.16 (strictly opt-in; must use O_DIRECT): LU-13805
  - autoIO (Hybrid I/O) for Lustre 2.16 and 2.16+
  - DLM lock contention detection: LU-12550
  - Server-side writeback and delayed alloc: LU-12916
  - Cross-file batching: LU-16355

- Future work
  - Modify autoIO decision by adopting ML based prediction to optimize I/O bandwidth;
  - Server-side algorithm which considers the server state

Whamcloud

Thank You!

DDN® STORAGE