Hewlett Packard Enterprise

MPI-IO Local Aggregation as Collective Buffering

Presenter: John Fragalla, HPE, Distinguished Technologist
Authors: Michael Moore, HPE, Master Technologist
Ashwin Reghunandanan, HPE, Systems/Software Engineer
Dr. Lisa Gerhardt, Lawrence Berkeley National Laboratory

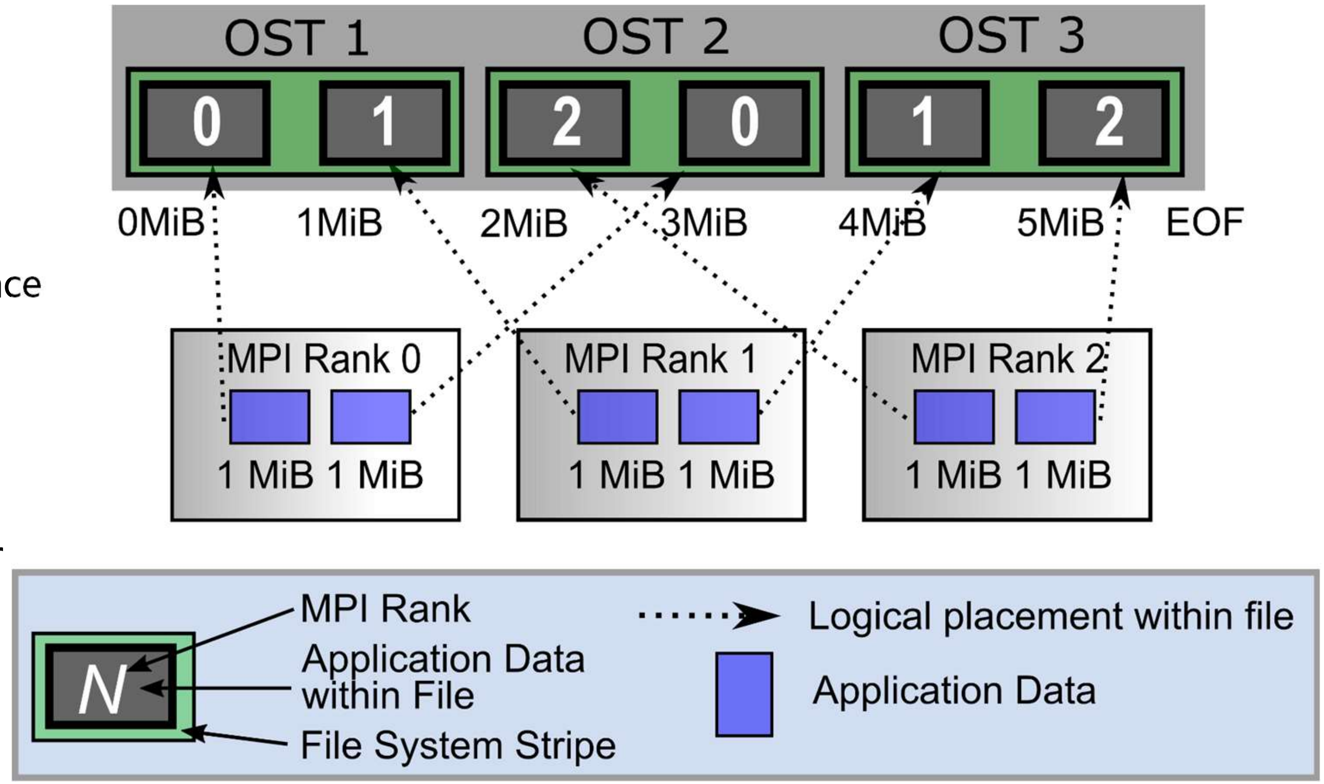
May 9, 2023

Agenda

- Background and Previous Work
- Collective MPI-IO Optimizations and Motivation
- Synthetic Benchmarks
- Local Aggregation as Collective Buffering Performance
- Using Cray MPICH MPI-IO statistics and timers
- General guidance on N:1 write performance optimization

Background: I/O Workloads

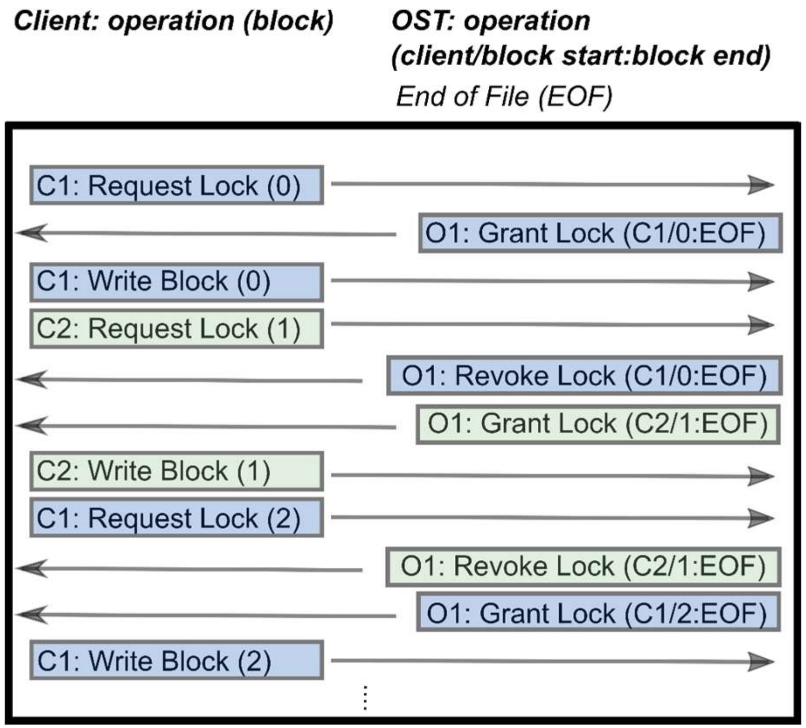
- File Per Process (N:N)
  - No write lock contention, optimal performance
- Shared File Workload (N:1)
  - More challenging for file systems since write locks can serialize access
  - Optimistically achieve performance equal to file per process
  - Segmented N:1 workload is the targeted workload for the remainder of the paper



2 MiB Lustre Stripe
IOR invocation: `IOR -a MPIIO -g -b 1m -t 1m -s 2 -o TESTFILE -w`

Background: File System Optimizations

- Default Lustre Lock Behavior
  - Optimized for File Per Process
  - Serializes writes to a single OST
- Lustre Group Locks
  - Defines a new type of lock that removes the file system doing byte-range locking (LDLM)
  - Application is responsible for data consistency
- Lustre Lock Ahead
  - Stops the default behavior of extending the lock grant to EOF on the specific OST
  - Nodes must request specific byte ranges of locks, which can happen asynchronously
- Lustre Overstriping
  - Allow multiple Lustre stripes per OST -- previously only one was allowed
  - Artificially increases the number of locking domains for a given number of OSTs



Default Lustre Locking behavior with two clients writing to non-overlapping offsets to the same OST

Background: Optimizations in the I/O stack

- POSIX API - Directly interact with the file system optimizations through ioctls (liblustre API calls)
- MPI-IO – POSIX optimizations aren't directly accessible to application
  - The MPI library uses them internally and applications can enable using them through MPI-IO Hints
- HDF5 – we're focusing on collective MPI-IO although HDF5 can use other APIs

API	Lustre Group Locks	Lustre Lock Ahead	Lustre Overstriping	Coll. MPI-IO with Coll. Buffering and Lustre Lockahead	Coll. MPI-IO with Coll. Buffering and Lustre Group Locks	Coll. MPI-IO with local aggregation as Coll. Buff.
POSIX						
Independent MPI-IO						
Coll. MPI-IO w/ Coll. Buff.						
HDF5 no Coll Metadata						
HDF5 with Coll Metadata						
Legend	Not Possible Due to Technical Limitations e.g. requires independent MPI-IO which isn't allowed with Lustre Group Locks			Not Applicable	Possible but requires application code changes	Available or Possible without code changes

Background: Collective MPI-IO and Motivation

- Many application I/O libraries (e.g. HDF5, NetCDF) use collective MPI-IO; that workload is our focus
- Brief Collective MPI-IO with Collective Buffering review:
  - With collective buffering a subset of MPI ranks are used to aggregate I/O requests made through a collective MPI-IO call
  - All MPI ranks in the communicator submit data to be written
  - A subset of MPI ranks, 'the aggregators', receive data from other MPI ranks and submit the I/O to the file system on their behalf via POSIX I/O calls coalescing data into larger requests
- Collective MPI-IO calls use collective buffering to:
  - Use POSIX accessible optimizations on the aggregators, like Lustre group locks or Lustre Lock ahead
  - Assign contiguous, Lustre stripe sized ranges to specific aggregators allowing one aggregator to only write to a single OST in Lustre stripe-sized chunks
  - Avoid read-modify-write operations for single MPI rank data that is less than a RAID stripe size (1 MiB on HPE Cray ClusterStor E1000)
  - Reduce the number of compute nodes writing to a single, shared file

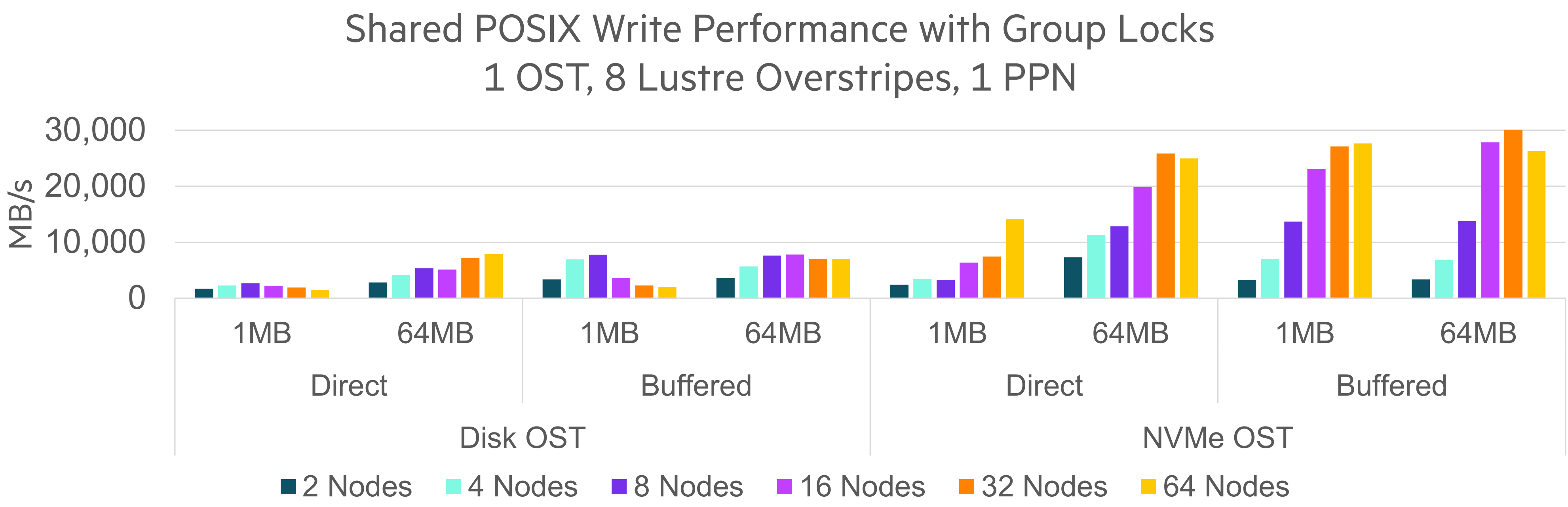
Background: Collective MPI-IO and Motivation

- Downsides of Collective Buffering
  - All the data passes over the fabric twice – one copy to the aggregators, another to get to the storage
  - OSTs have changed over time
    - Historically: basic RAID-6 OSTs could achieve 1-2 GB/s allowing a single aggregator MPI rank to achieve peak performance
    - Now:
      - OSTs have more drives (106 drives or 24 NVMe instead of 8+2)
      - OSTs use high performing, non-rotational media
      - A single OST can be 5x - 40x the performance of older OSTs and require 8 or more aggregators to achieve peak performance
  - Aggregators, for collective buffering, are just application MPI ranks
    - No dedicated memory for buffering larger quantities of data (tens or hundreds of GBs)
    - For large collective writes many iterations of receiving and writing data are required
- For many collective MPI-IO workloads collective buffering was just a pass-through that allowed using optimized locks
- How do we keep the locking benefits of collective buffering without the bottleneck of aggregators?

Collective MPI-IO Local Aggregation as Collective Buffering

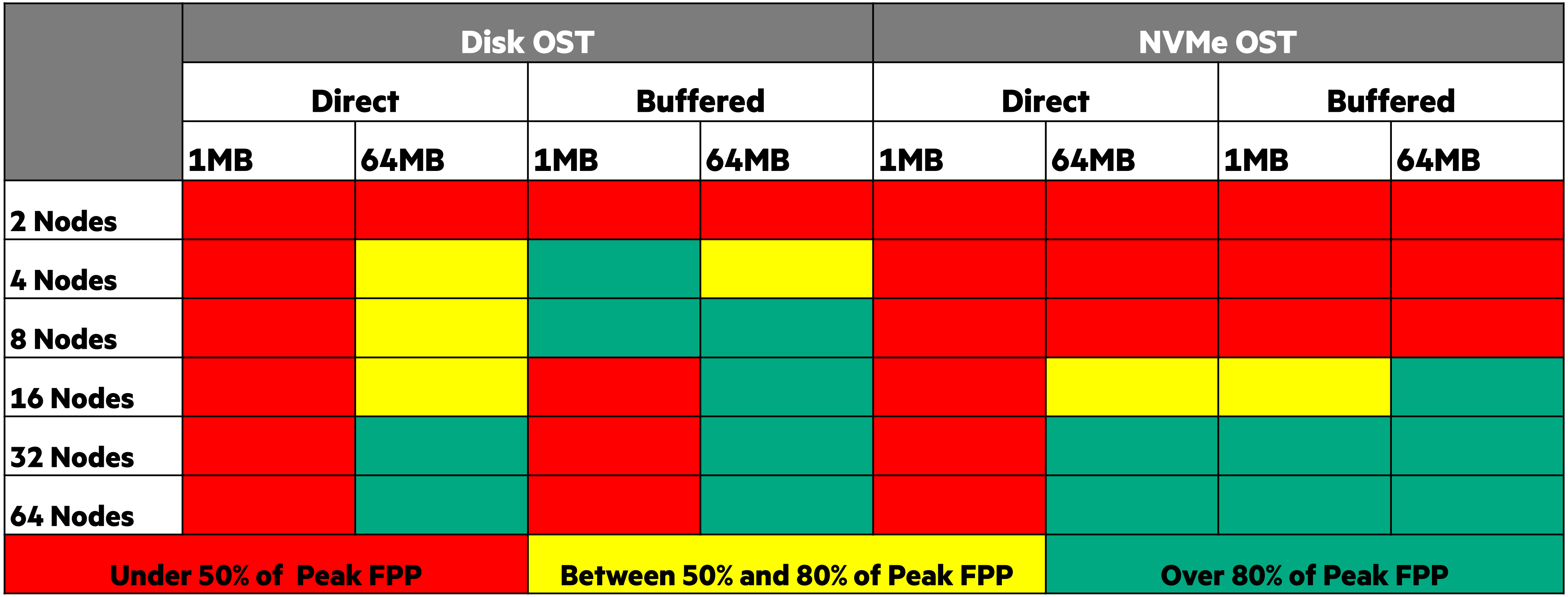
- Extend the idea of an aggregator so every node serves as its own aggregator
  - Avoid the double copy of data over the fabric by each node writing its own data
  - Each node writing data (all nodes in the MPI communicator) take a Lustre group lock to use optimized locking
  - Initial implementation has each rank performing its own I/O
- Higher performance at lower node counts today is limited:
  - Shared file write performance is limited to 1-2 GB/s for any buffered I/O writes regardless of PPN
  - Current Collective Buffering implementation is optimized for a single rank per node as an aggregator
  - Using 32 to 64 nodes per NVMe OST to drive a high percentage of peak performance requires at or above full system scale for many HPC systems
- Downside of this approach
  - There is no guarantee of stripe aligned accesses as in the case of collective buffering
  - With group locks the impact of this should be minimal

Performance: Current HPE Cray ClusterStor Shared Write



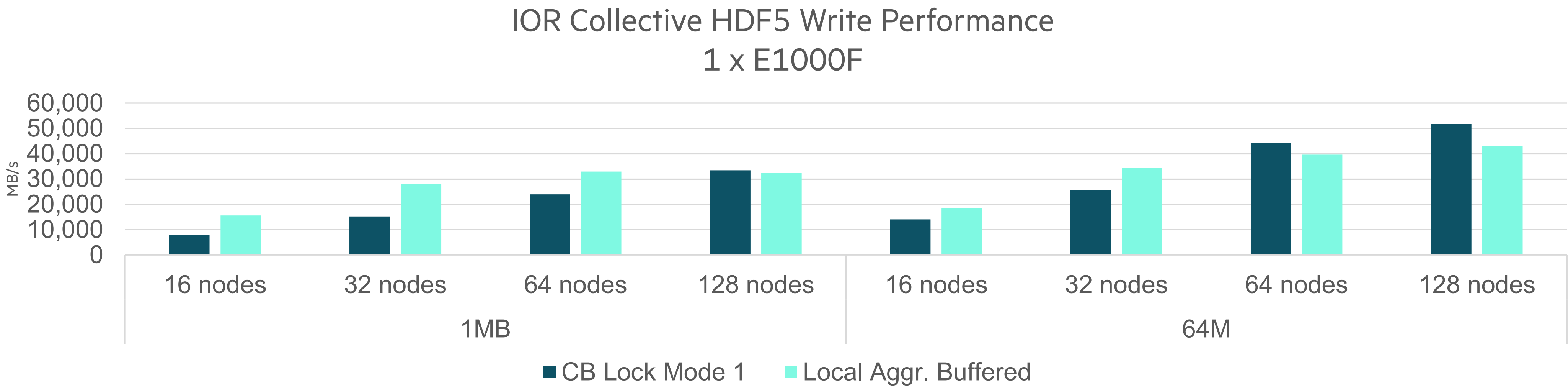
- I/O request size effects how many nodes are needed to achieve peak performance
- Direct I/O is not effective at smaller transfer sizes but is more beneficial with multiple processes per node
- Effectively measuring collective buffering with multiple aggregators per OST

Performance: Shared Write relative to File Per Process Write



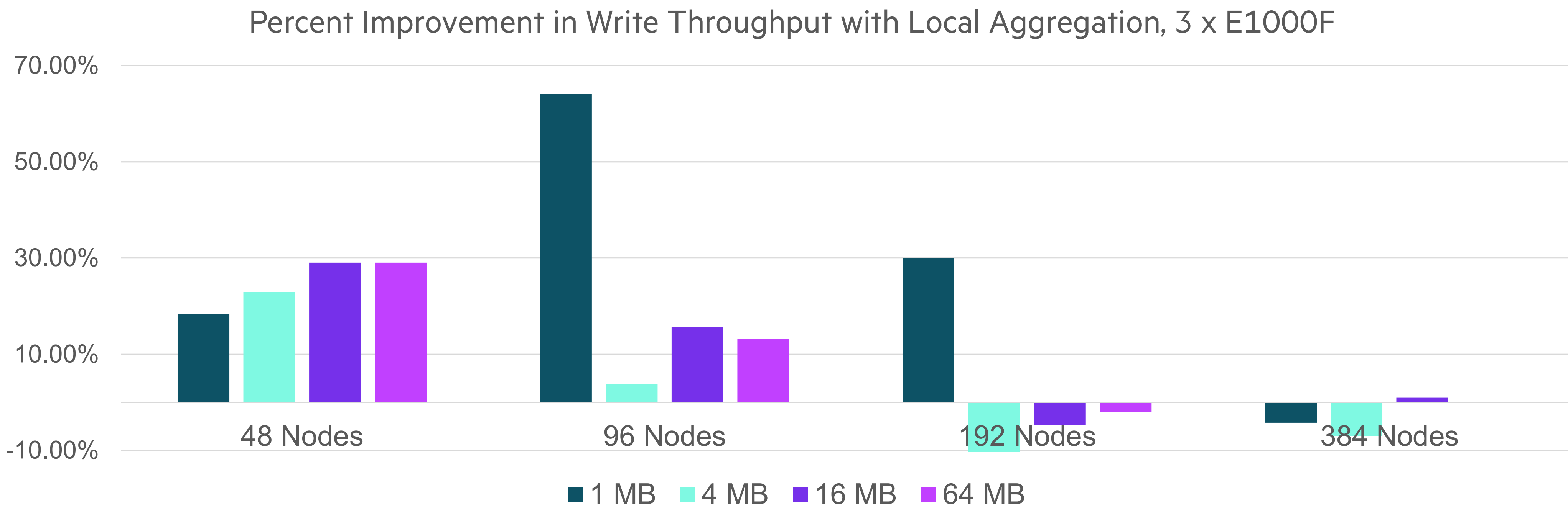
Single OST, 1 PPN (i.e. a collective buffering aggregator workload)

Performance: HDF5 IOR Benchmarks



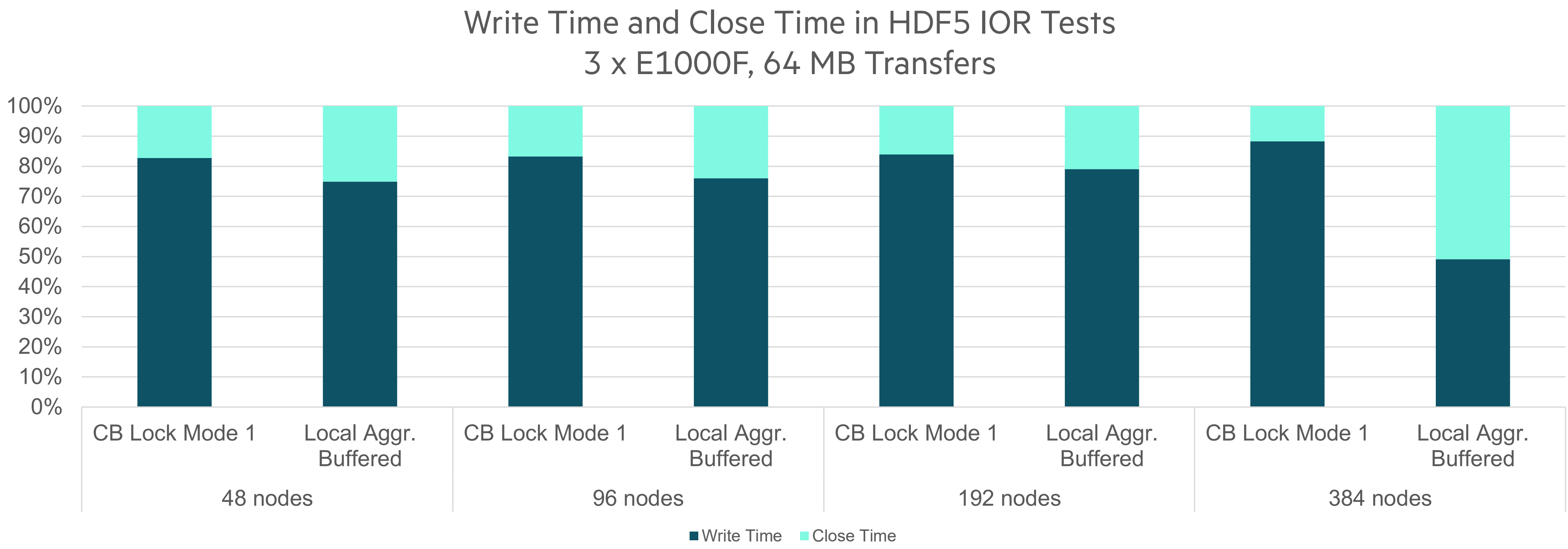
- Significant improvement at lower nodes counts (relative to number of OSTs) and smaller transfer sizes
- Not always advantageous although close time issue makes direct comparison difficult
- IOR 3.3 does not seem to correctly align HDF5 accesses even using the '-J' parameter
  - This prevented using direct I/O since accesses were not 4k aligned

Performance: HDF5 IOR Benchmarks



- Scaling up the OST count and maintaining the same ratio of nodes to OSTs continues to show meaningful improvement using Local Aggregation as Collective Buffering
- At higher node counts per OST the improvement seems to lessen but a majority of the performance gains are lost to higher close time since more nodes open the file in the Local Aggregation as Collective Buffering case

Performance: HDF5 IOR Benchmarks, Close Time Issue



- LU-16046 increases time for Group Lock Unlock; a fix will be in COS 2.6 Lustre clients – this reduces the measured throughput but effects all jobs taking group locks in a similar way
- There may be a second issue as longer close times, not group unlock times, were sometimes reported
- This disproportionately reduces Local Aggregation as Collective Buffering results, especially at larger node counts.

Usage: How to enable the Local Aggregation as Collective Buffering

- Standard Collective Buffering with Lustre Group Locks
  - ```
    MPICH_MPIIO_HINTS="*:romio_cb_write=enable:cray_cb_nodes_multiplier=4:\
                       cray_cb_write_lock_mode=1:romio_no_indep_rw=true"
    ```
  - cray_cb_nodes_multiplier is the number of aggregators per Lustre stripe (not Lustre OST)
    - This is the number of nodes with 1 PPN in the previous shared file test results
- Local Aggregation as Collective Buffering MPI-IO hints
  - ```
    MPICH_MPIIO_HINTS="*:romio_cb_write=disable:cray_nocb_write_lock_mode=1:\
                      romio_cb_read=disable:romio_ds_write=disable:\
                      romio_no_indep_rw=true"
    ```

Analysis: Cray MPICH MPI-IO Info and Statistics

- Cray MPICH includes several environment variables to report out information on MPI-IO activity

```
export MPICH_MPIIO_HINTS_DISPLAY=1
export MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1
export MPICH_MPIIO_TIMERS=1
export MPICH_MPIIO_STATS=1
```

- MPICH_MPIIO_HINTS_DISPLAY
  - Confirm which hints are in effect for a given file in case there are conflicting options, incorrect wildcard, etc.
  - A filtered list of relevant hints

```
PE 0: MPIIO hints for /lus/flash/testdir.1502/IOR:
          romio_cb_read             = enable
          romio_cb_write            = enable
          romio_no_indep_rw         = true
          cray_cb_nodes_multiplier = 4
          cray_cb_write_lock_mode  = 1
          cb_nodes                  = 8
```

Analysis: Cray MPICH MPI-IO Info and Statistics

- MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY
  - Show which MPI ranks and nodes were serving as aggregator ranks

```
Aggregator Placement for /lus/flash/testdir.205/IOR
 RankReorderMethod=1  AggPlacementStride=-1
 AGG    Rank       nid
 ----  ------  --------
    0       0  nid00000
    1     256  nid00008
    2     512  nid00016
    3     768  nid00024
    4    1024  nid00032
    5    1280  nid00040
    6    1536  nid00048
    7    1792  nid00056
```

Analysis: Cray MPICH MPI-IO Info and Statistics

- MPICH_MPIIO_STATS=1
  - Display counts and sizes of MPI-IO operations

```
+--------------------------------------------------------+
| MPIIO write access patterns for /lus/flash/testdir.205/IOR
|   ranks in communicator   = 2048
|   independent writes      = 0
|   collective writes       = 64
|   independent writers     = 0
|   aggregators             = 8
|   stripe count            = 2
|   stripe size             = 16777216
|   system writes           = 131072
```

Analysis: Cray MPICH MPI-IO Info and Statistics

- MPICH_MPIIO_STATS=1
  - Aggregators active is specific to collective buffering enabled

```
|   stripe sized writes     = 131072
|   aggregators active      = 0,0,0,131072 (1, <= 4, > 4, 8)
|   total bytes for writes  = 2199023255552 = 2097152 MiB = 2048 GiB
|   ave system write size   = 16777216
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 0
|   ave write gap size      = NA
+----------------------------------------------------------+
```

Analysis: Cray MPICH MPI-IO Info and Statistics

- MPICH_MPIIO_TIMERS=1
  - Shows timing (min, max, avg) for the different phases of collective MPI-IO
  - Reports throughput based on reported timings for different phases
  - Always reports an 'all ranks' section for write or read. When using collective buffering a

```
+--------------------------------------------------------------------------+
| MPIIO write by phases, all ranks, for /lus/flash/testdir.205/IOR
|   number of ranks writing          =      8
|   number of ranks not writing      =   2040
|                                             min          max          ave
|                                      ----------   ----------   ----------
|   open/trunc  time                 =       0.01         0.01         0.01
|   close sys time                   =       0.00         0.00         0.00
|   close fsync time                 =       0.00         0.00         0.00
|   close group-unlock time          =       0.00        55.53         0.21
|   close other + wait time          =       0.00         2.82         0.00
```

Analysis: MPICH_MPIIO_TIMERS=1

- Several counters are reported in ticks; you can convert them to seconds if you want absolute time
  - Calculate the tics_per_second based on a reported time and time scale that's reported like file write max:
  - Ticks_per_sec = (file write max) * (time scale) / file write max seconds e.g. `(490903754)*(2^10) / (229.23)`
- Bandwidth report in 'all ranks' section

```
|   data send BW (MiB/s)          =                         17323.854
|   raw write BW (MiB/s)          =                          9433.878
|   net write BW (MiB/s)          =                          8381.648
```

- Timing in "Writers only" (aggregators) section

```
|   file write       time         =     215.93      229.23      222.30
|
|   time scale: 1 = 2**10     clock ticks     min         max         ave
|                                 ----------  ----------  ---------- ---
|   total                         =                         535355066
|   wait for coll                 =    2057775    32745487    10641530  1%
|   data send                     =   38484553    59658902    46869207  8%
|   file write                    =  462411061   490903754   476056748 88%
```

Analysis: MPICH_MPIIO_TIMERS=1

- The reported bandwidth are helpful but can be optimistic if taken as printed
  - In the previous example
    - MPICH_MPIIO_TIMERS output reports a write time of 229.23 seconds and net write BW of 8,381 MiB/s
    - IOR reports a write time of 224.26 seconds or 9.13 GiB/s
    - IOR reports the test throughput as 7.6 GiB/s
  - Application I/O time includes all the operations potentially including opening the file, flushing dirty data, releasing locks, and closing the file
  - MPICH_MPIIO_TIMERS output provides timers to understand where the other time is spent

Recommendations: Shared File Parameters

- Lustre Stripe Count
  - More frequently decided by job size and not file size
  - Consider potential OST performance
  - Traditional buffered shared file or collective buffering can achieve between 1.3- 1.7 GB/s write
  - Disk OST:
- Lustre Stripe Size
  - Larger transfer sizes generally do better all things equal
    - –Diminishing returns for stripe sizes larger than 64M
    - –Remember the *Lustre stripe size * Number of Aggregators* is the size of data that needs to be written in a collective write to use all aggreagors
  - Align the stripe size with the amount of data each node writes
    - –Example: if MPI rank writes 1MB, with 64 PPN, with a block rank allocation, a Lustre stripe size of 64M would allow a 64M contiguous request to be written – assuming offsets are stripe aligned and buffered I/O is used

Recommendations: Shared File Settings By API

- POSIX / Independent MPI-IO
  - Lustre Overstriping
  - Direct IO if multiple ranks writing and need more than 1.5 GB/s per node
  - Align Lustre stripe size with (*PPN * I/O request size*) block allocation

- Mixed Collective and Independent I/O (HDF5 without collective metadata)

```
MPICH_MPIIO_HINTS="*:romio_cb_write=enable:cray_cb_nodes_multiplier=N:\
                    cray_cb_write_lock_mode=2"
```

  – Lustre Overstriping did not show measurable performance improvements in NVMe OST testing

- Collective MPI-IO on NVMe OSTs
  - Current

```
MPICH_MPIIO_HINTS="*:romio_cb_write=enable:cray_cb_nodes_multiplier=64:\
                    cray_cb_write_lock_mode=1:romio_no_indep_rw=true"
```

  - Experimental Local Aggregation

```
MPICH_MPIIO_HINTS="*:romio_cb_write=disable:cray_nocb_write_lock_mode=1:romio_cb_read=disable:\
                    romio_ds_write=disable:romio_no_indep_rw=true"
```

- Collective MPI-IO on Disk OSTs

```
MPICH_MPIIO_HINTS="*:romio_cb_write=enable:cray_cb_nodes_multiplier=8:\
                    cray_cb_write_lock_mode=1:romio_no_indep_rw=true"
```

Future Work

- Get LU-16046 fix for group unlock duration into shipping Lustre client code
  - Planned inclusion in COS release (2.6), potential for backports to older COS versions but not plan of record
- Investigate data validation and close time issues
- Configurable number of ranks per node acting as aggregators user intra-node transfers
  - Currently all ranks submit their own I/O requests
- Optimize collective buffering behavior for many aggregators per node
  - Currently single rank per node is the optimal configuration which requires more nodes to get peak performance
- Larger scale and application testing

Summary

- Higher performance NVMe OSTs created challenges for existing collective MPI-IO optimizations
  - The extra data copy adds latency and many workloads only make use of locking optimizations through coll. buff.
  - Very high node counts per OST are needed to achieve peak performance which isn't always feasible

- A new, experimental, feature was added to HPE Cray MPICH: Local Aggregation as Collective Buffering
  - Optimizes Shared file writes with Collective MPI-IO by:
    - Removing the overhead of sending data to aggregator ranks (collective buffering)
    - Retains the use of Lustre Group Locks for optimized POSIX file writes

- HPE Cray MPICH provides several collective MPI-IO debugging options to understand Collective MPI-IO performance, helping to optimize your application's workload for your storage environment

- Using the new Local Aggregation as Collective Buffering feature shows significant improvements in moderate scale testing of over 60% for some workloads

Thank you

Michael Moore <michael.moore@hpe.com>
Ashwin Reghunandanan <ashwin.reghunandanan@hpe.com>
Lisa Gerhardt <lgerhardt@lbl.gov>