Guidelines for Efficient Parallel I/O on the Cray XT3/XT4

Jeff Larkin, Cray Inc.

Mark Fahey, ORNL

Overview

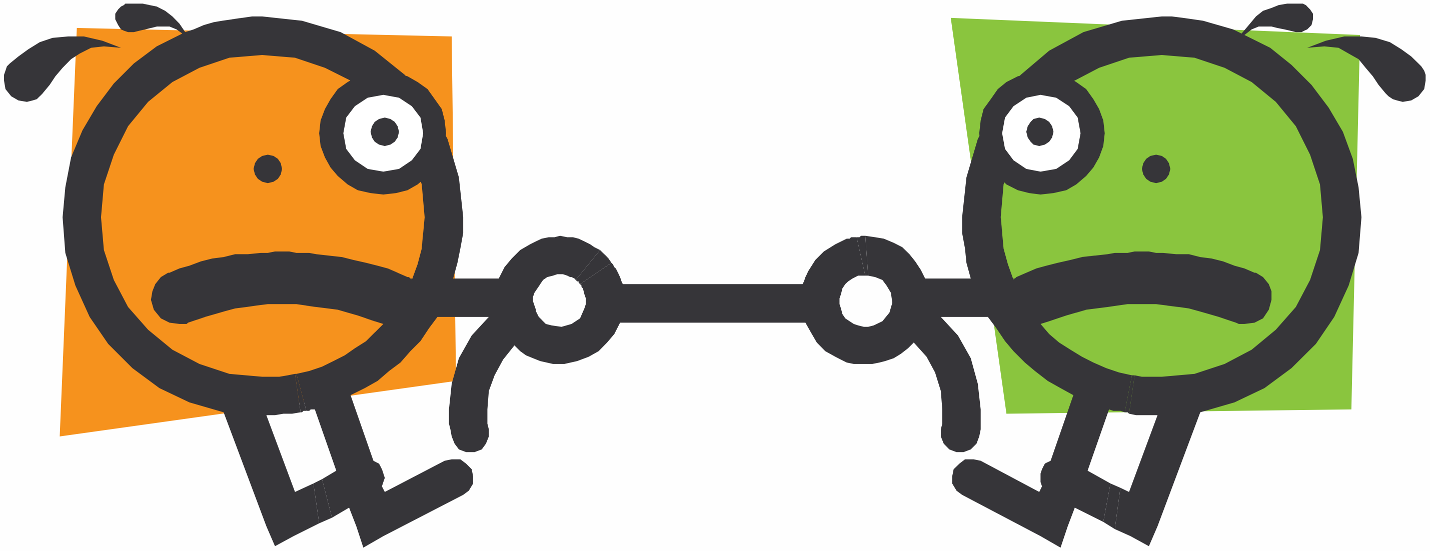
- What's the problem?
- “Typical” Application I/O
- A Possible Solution
- Benchmark Results
- Suggested Guidelines
- Future Work

What's The Problem?

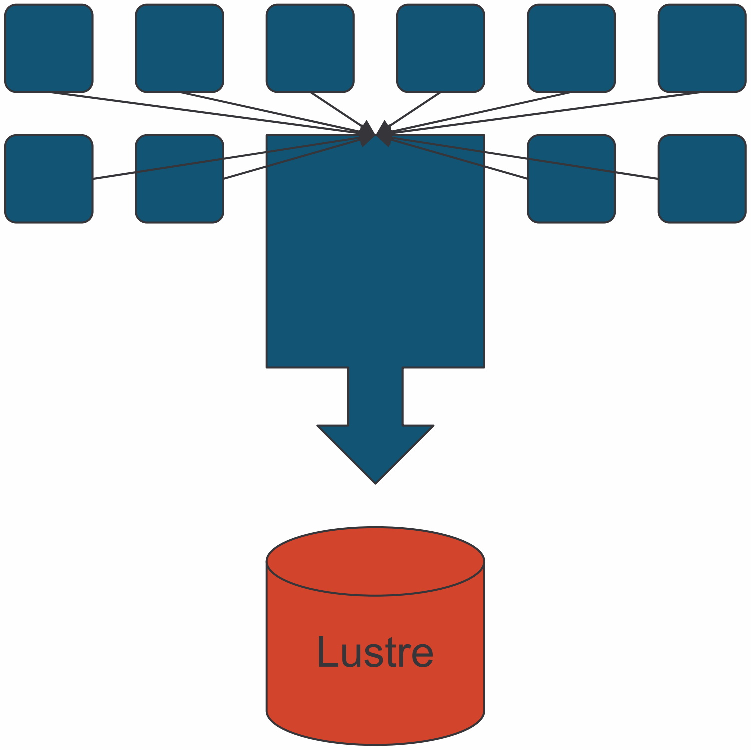
- Flops are Cheap, Bandwidth isn’t
- Machines and Applications aren’t getting any smaller
- But...
    - Isn't Lustre enough?
    - Can't I use libraries?
    - Doesn't it just work?
- Without user or programmer intervention, I/O will not perform at practical peak
- There is no *Silver Bullet*

"Typical" Application I/O

- THERE IS NO TYPICAL APPLICATION I/O
- There are several common methods, but 2 are very common and problematic
  - Single-writer reduction
  - N-writer/N-reader to N-files



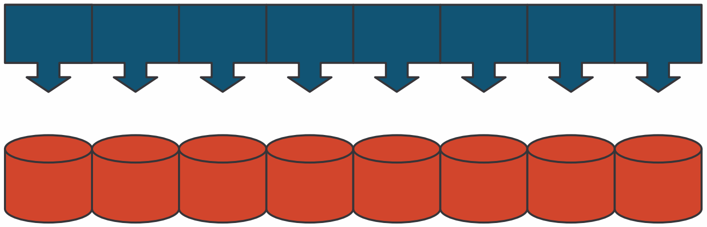
Single-writer Reduction



- The Plan
  - All processors send to 1 I/O node for output
  - File striped to maximum OSTs
- The Problem
  - Even with maximum striping, 1 node will never achieve maximum bandwidth
  - single node IO bandwidth is extremely limited
  - reading/writing a terabyte would require more than 1 hour at current I/O rates

N-Writer to N-Files

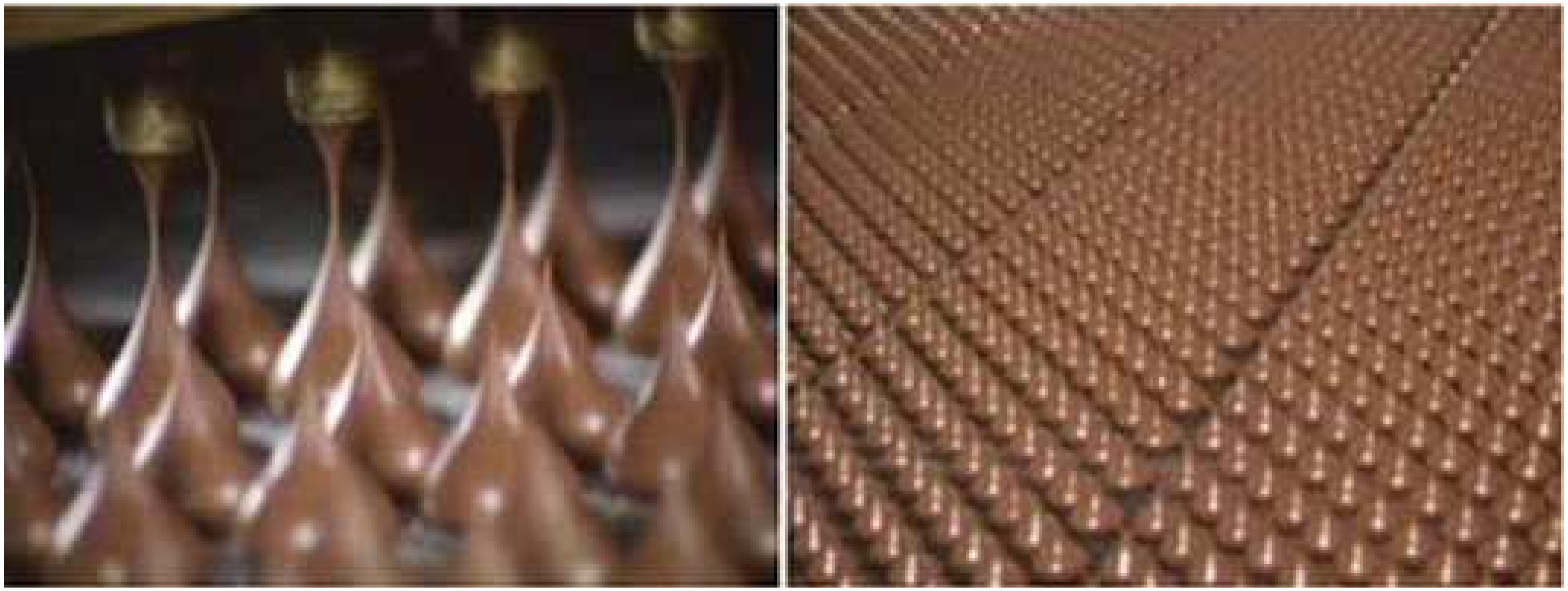
- The Plan
  - Every process opens a file and dumps its data
  - Files striped to 1 OST
- The Problem
  - Can lead to slow opens and general filesystem slowness
  - If the writes are not large, performance will suffer
  - Inconvenient
- One Modification
  - Use MPI-I/O for just 1 file
  - Suffers when i/o results in small buffers
  - Data shows is as non-optimal



The Problem

- **These methods are fine until you scale-up**
  - Proof forthcoming
- **Without user-intervention you will not get practical peak I/O bandwidth**
- **For example**
  - Single writer/reader reduction
    - Even with maximum striping on file, effective bandwidth is limited by the 1 compute node
  - All processes read/write at the same time
    - Slow metadata operations (all hit the MDS at the same time)
    - Overwhelm OSTs and/or IO service nodes
    - Possibly inconvenient to users

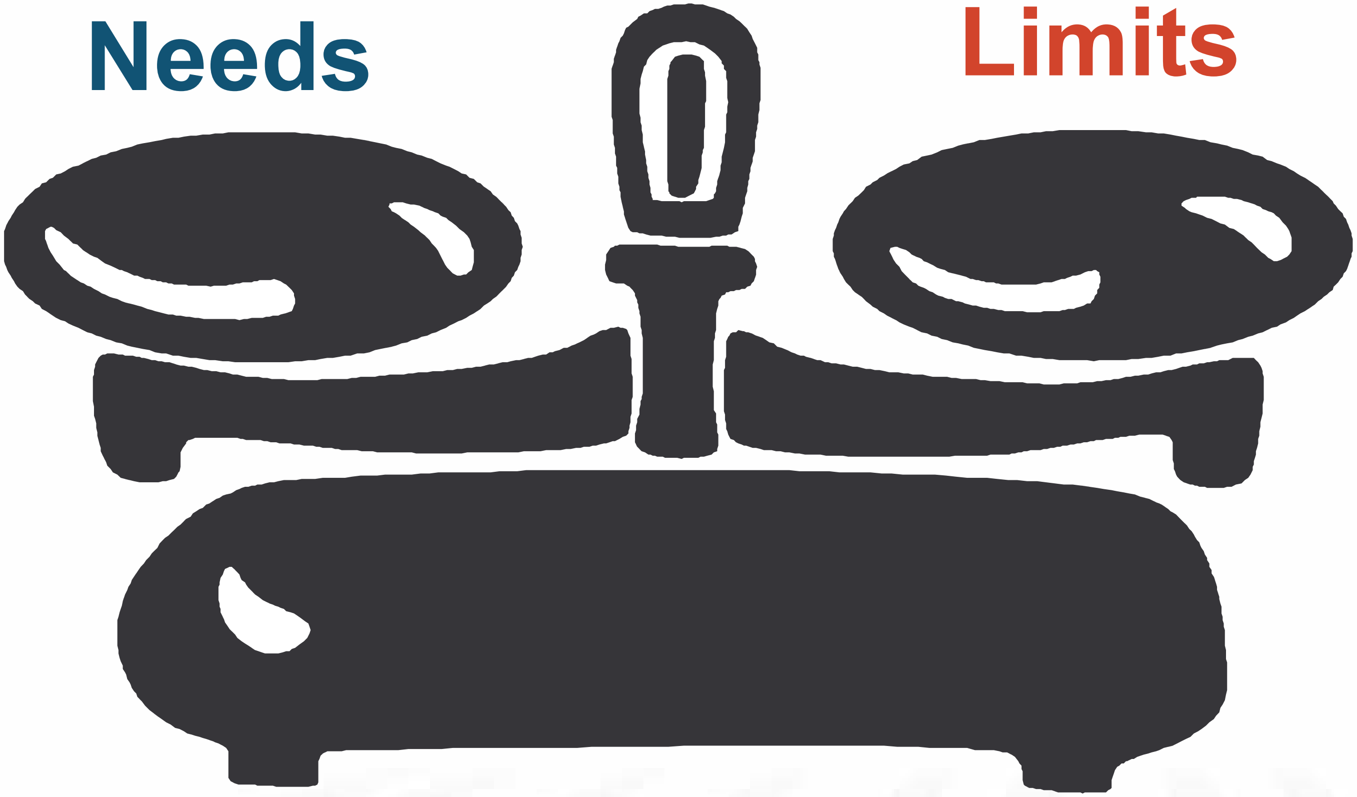
What does efficient I/O look like?



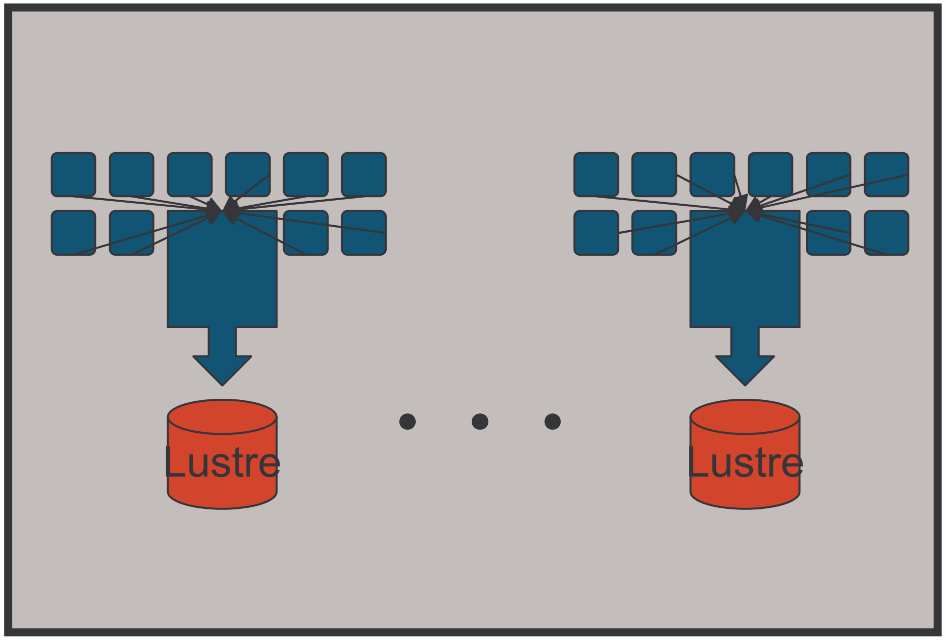
Striking a Balance

Application Needs

Filesystem Limits



Subsetting Approach



- The Plan:
  - Combine the best of our first two I/O methods
  - Choose a subset of nodes to do I/O
  - Send output to or Receive input from 1 node in your subset
- The Benefits
  - High Bandwidth, Low FS Stress
  - May improve I/O Buffering
- The Costs
  - I/O Nodes must sacrifice memory for buffer
  - Requires Code Changes

Subsetting (cont.)

- **Assumes job runs on thousands of nodes**
- **Assumes job needs to do large I/O**
- **From data partitioning, identify groups of nodes such that:**
  - each node belongs to a single group
  - data in each group is contiguous on disk
  - each group has enough data to sufficiently buffer
  - there should be at least as many groups as OSTs you plan to use
- **Pick one node from each group to be the ionode**
- **Use MPI to transfer data within a group to its ionode**
- **Each IO node reads from/writes to disk**

Example code

create an MPI communicator that include only ionodes; `listofionodes` is an array of the ranks of writers/readers

```
call MPI_COMM_GROUP(MPI_COMM_WORLD, &
   WORLD_GROUP,ierr)

call MPI_GROUP_INCL(WORLD_GROUP,nionodes, &
  listofionodes,IO_GROUP,ierr)

call MPI_COMM_CREATE(MPI_COMM_WORLD,IO_GROUP, &
  MPI_COMM_IO,ierr)
```

Example code (cont.)

open

```
call MPI_FILE_OPEN(MPI_COMM_IO, trim(filename), &
     filemode, finfo, mpifh, ierr)
```

read/write

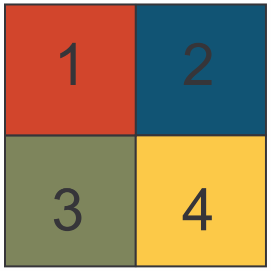
```
call MPI_FILE_WRITE_AT(mpifh, offset, iobuf, &
      bufsize, MPI_REAL8, status, ierr)
    OR
call MPI_FILE_SET_VIEW(mpifh, disp, MPI_REAL8, &
      MPI_REAL8, "native", MPI_INFO_NULL, ierr)
call MPI_FILE_WRITE_ALL(mpifh, bigA, size(bigA), &
      MPI_REAL8, status, ierr)
```

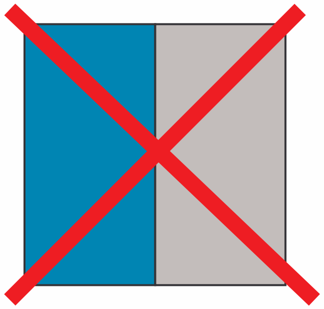
close

```
call MPI_FILE_CLOSE(mpifh, ierr)
```

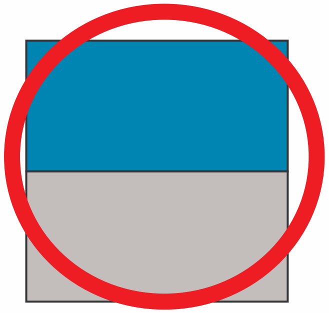
Sample Paritioning: POP

- data is 3d - X, Y, Z
- X and Y dimensions are partitioned in blocks
- sample 4 node partition:
  - Each of the 4 colored blocks represents one node’s part of the data
  - Each of the two lighter colored blocks represent 1 I/O Node
  - I/O Groups should be arranged so their data is contiguous on disk



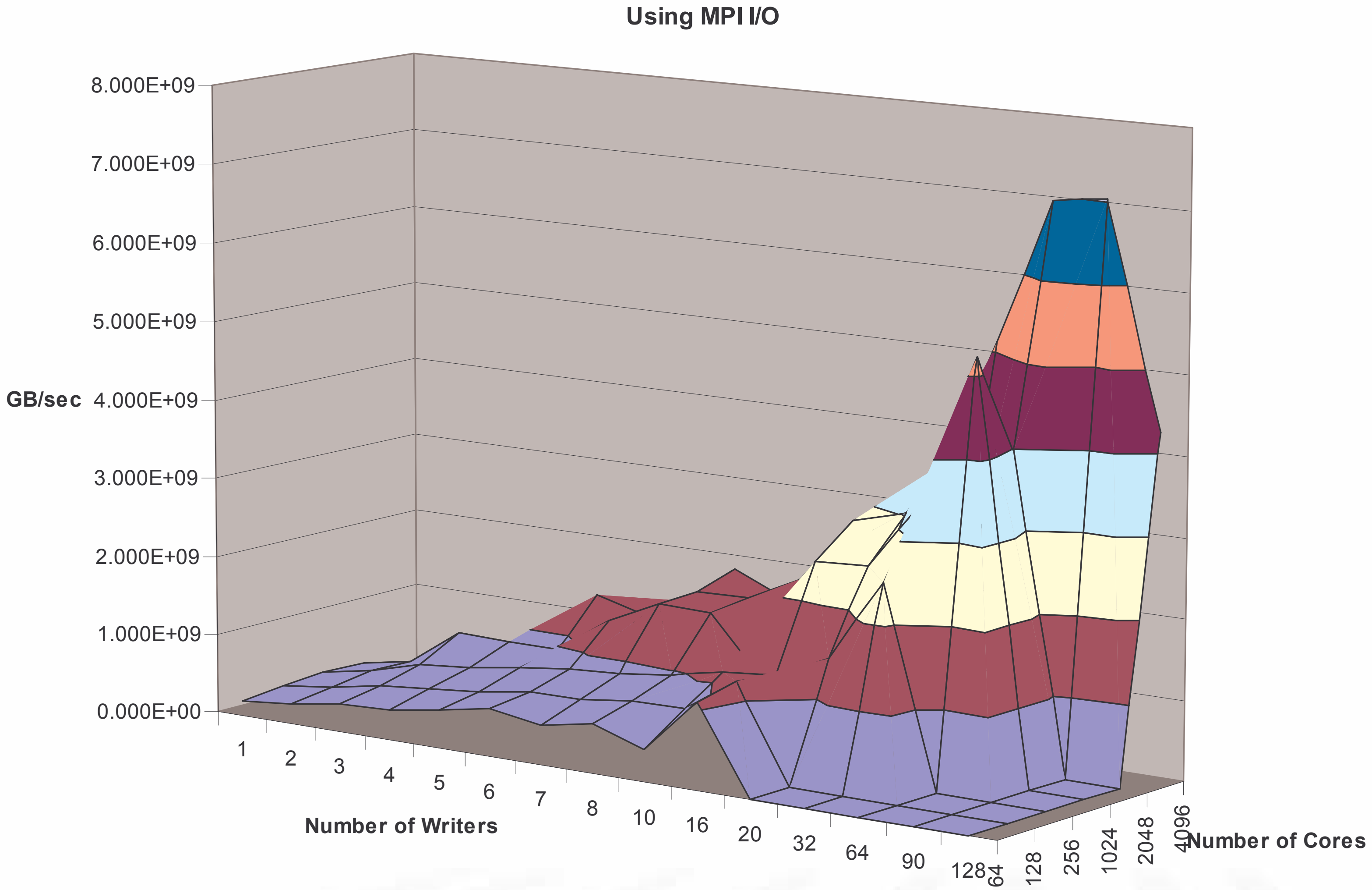


Data from nodes 1 & 3 alternate on disk. This will perform slowly and can’t adjust to more processors.



Data from node 1 is contiguous, followed by data from node 2, which is also contiguous.

A Subset of Writers Benchmark



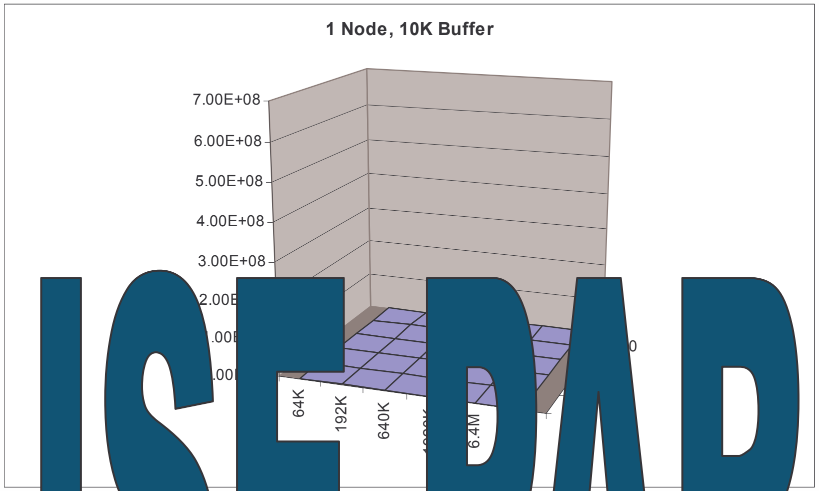
Caveats

- **OS level and Lustre configuration not consistent for all tests**
  - Striping tests done with 1.5.25
  - Some with 1.5.29 and others with 1.5.31
- **Some results from XT3 and some from XT4**
- **Some runs done in dedicated mode**
- **And others done during regular production usage**
  - For these, we report the “max” time over many trials - sort of a practical peak
- **Since we are looking at trend, not specific numbers, these caveats are not a problem**

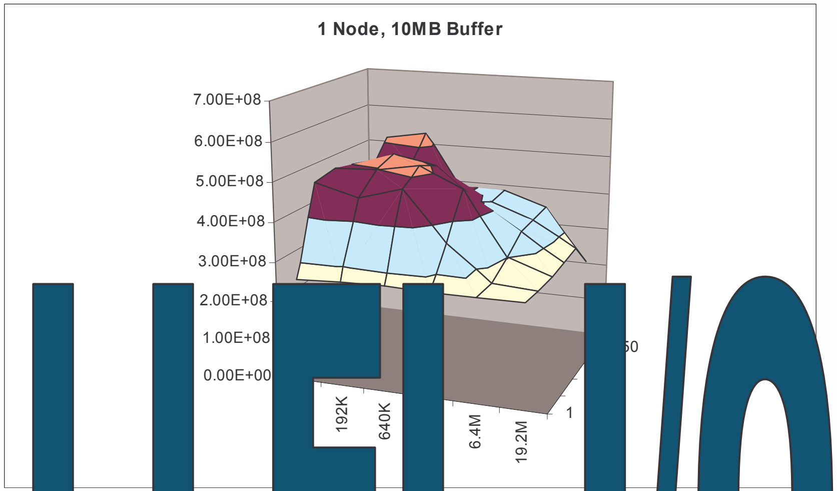
Benchmark 1 Results: Things to Know

- Uses write_at rather than file partitioning
- Only write data...sorry
  - Read data was largely similar
- Initial benchmarking showed MPI transfers to be marginal, so they were excluded in later benchmarking

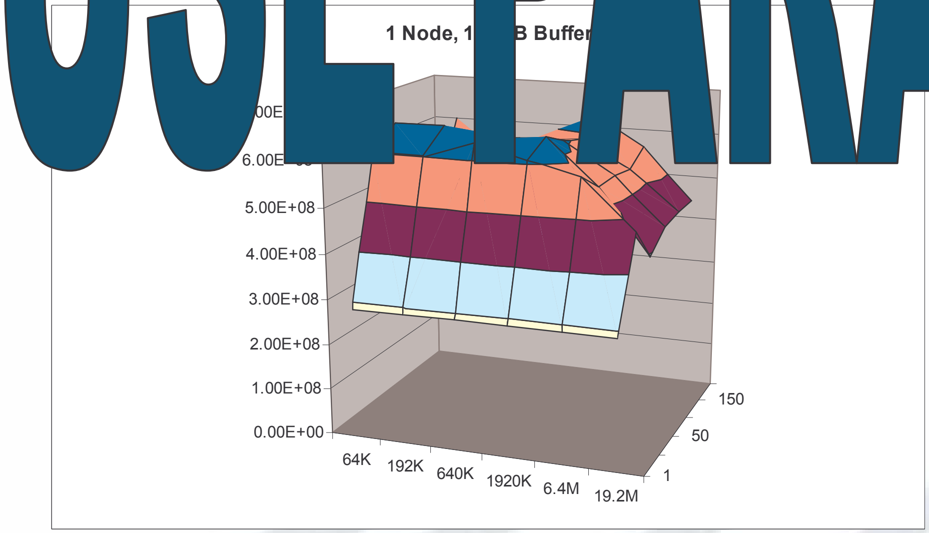
Benchmark 1 Results: 1 I/O Node - Stripes



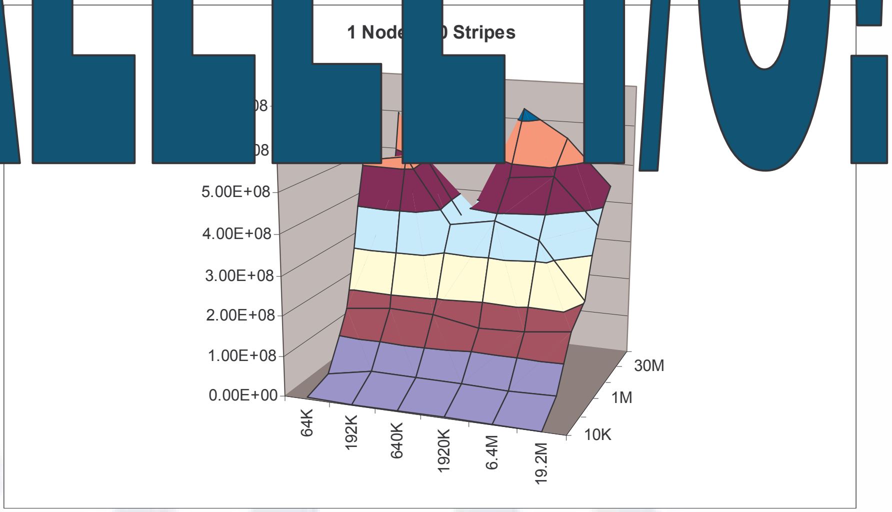
1 Node, 10K Buffer



1 Node, 10MB Buffer



1 Node, 1 B Buffer

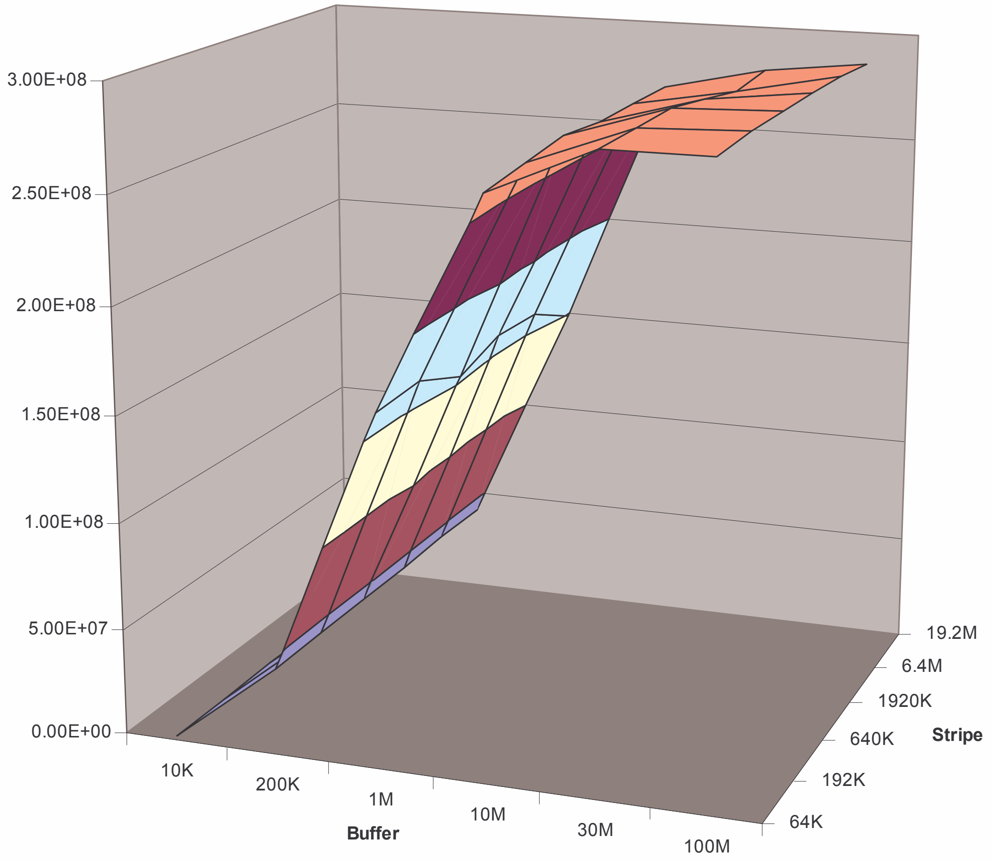


1 Node 0 Stripes

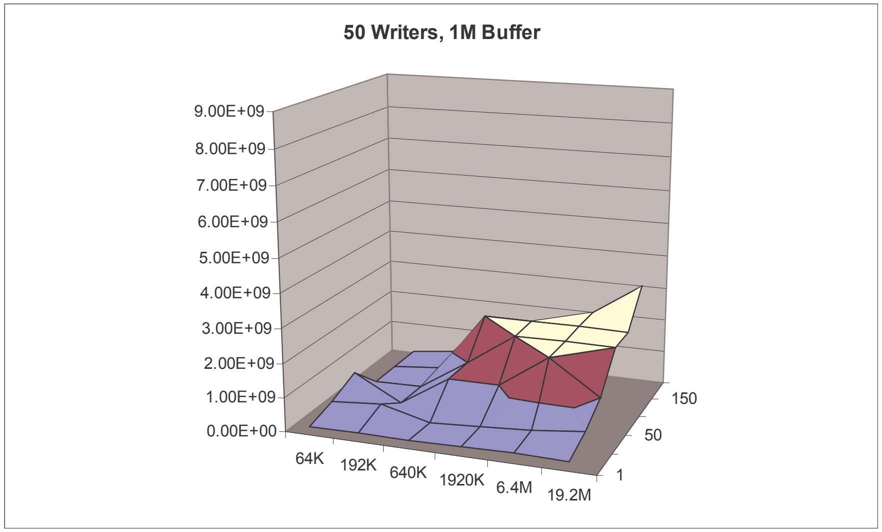
USE PARALLEL I/O!

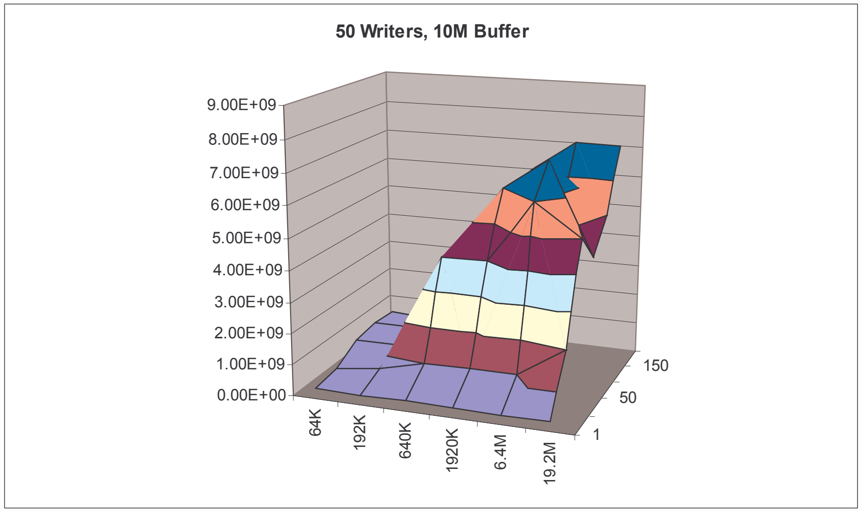
Benchmark 1 Results: 1 I/O Node – Buffer Size

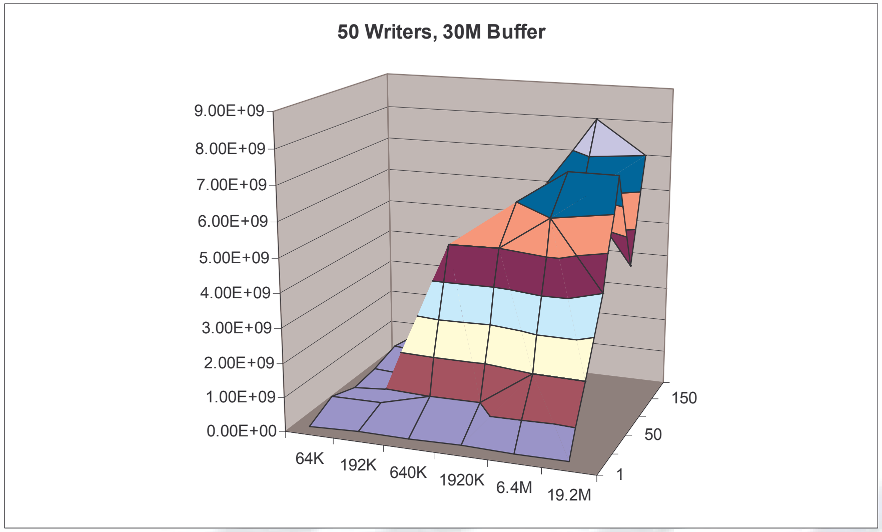
- Single node, single stripe: bandwidth of IO write to disk for different buffer sizes
  - Buffer size is the size of contiguous memory on one IO node written to disk with one write
- Buffer size should be at least 10 megabytes

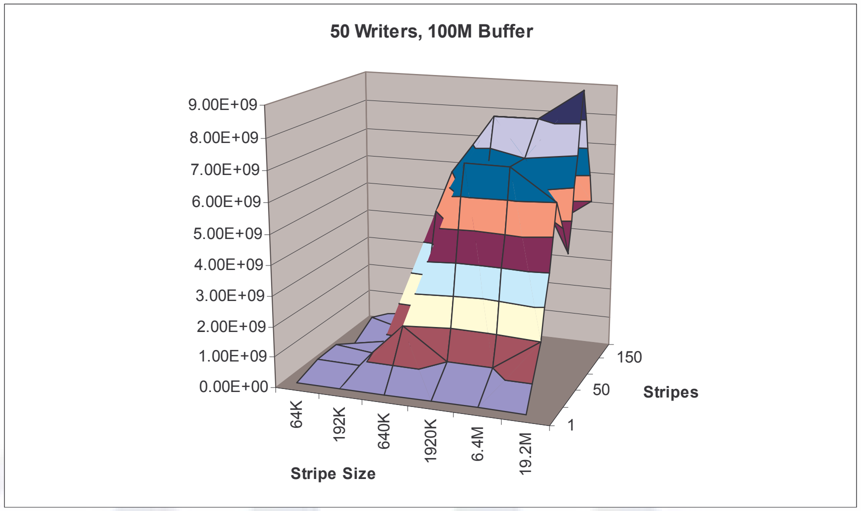


Benchmark 1: 50 Writers, Varying Stripe Count, Size and Buffer Size

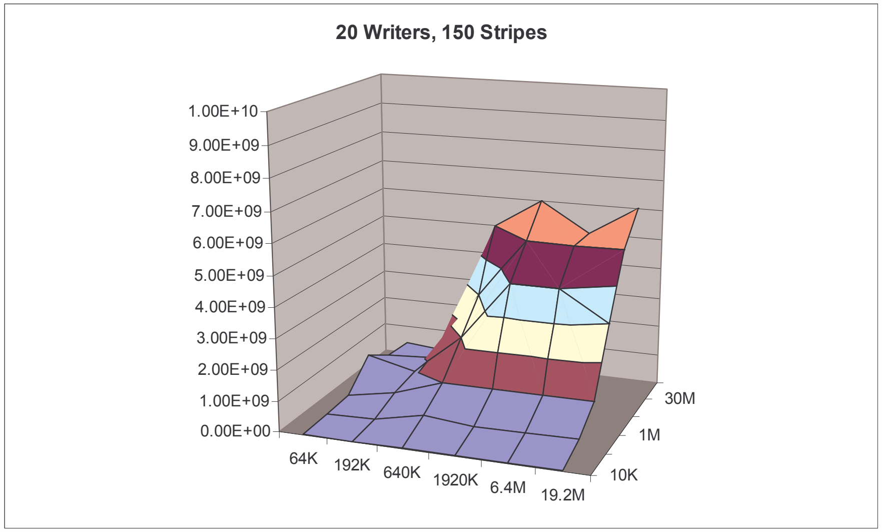

50 Writers, 1M Buffer

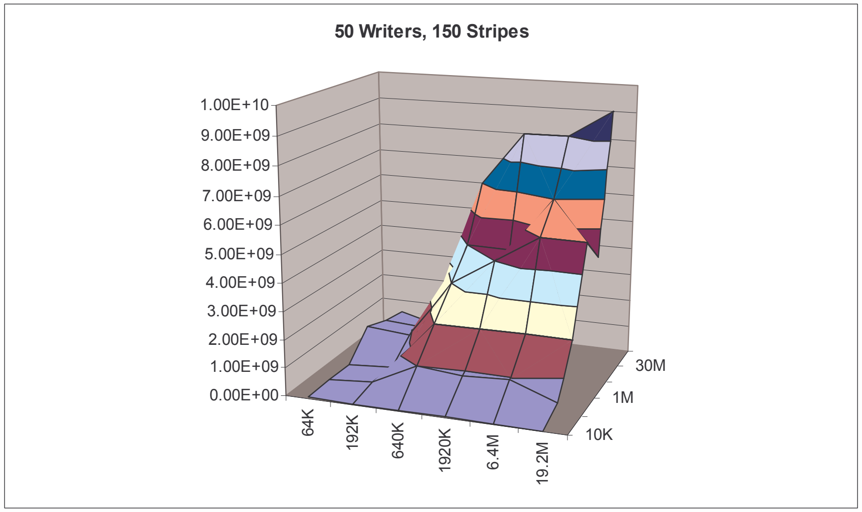

50 Writers, 10M Buffer

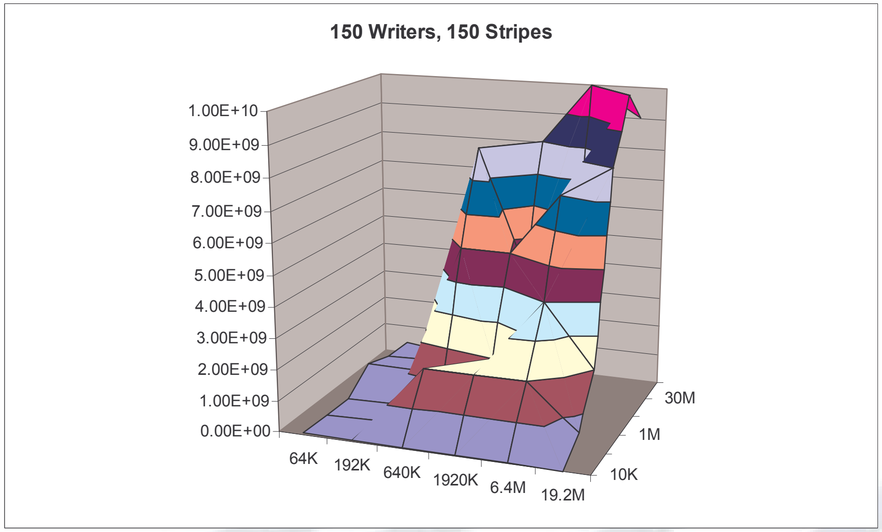

50 Writers, 30M Buffer

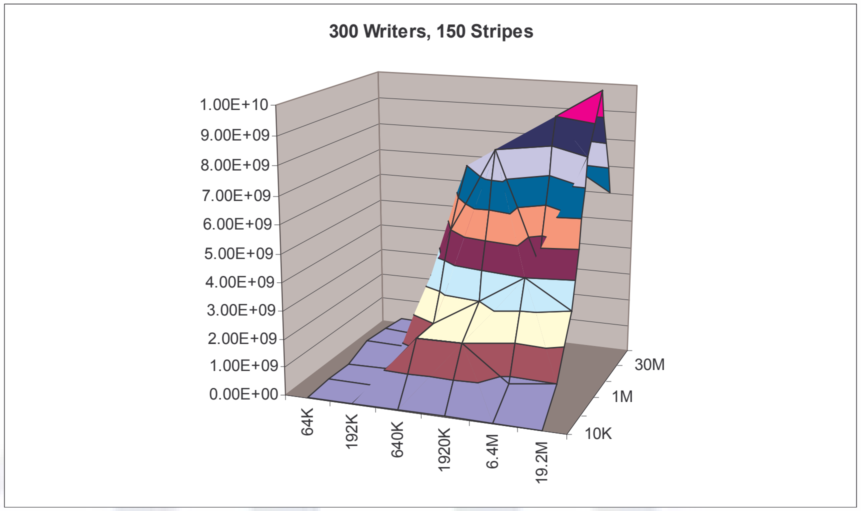

50 Writers, 100M Buffer

Benchmark 1: 150 Stripes, Varying Writers, Buffer, and Stripe Sizes

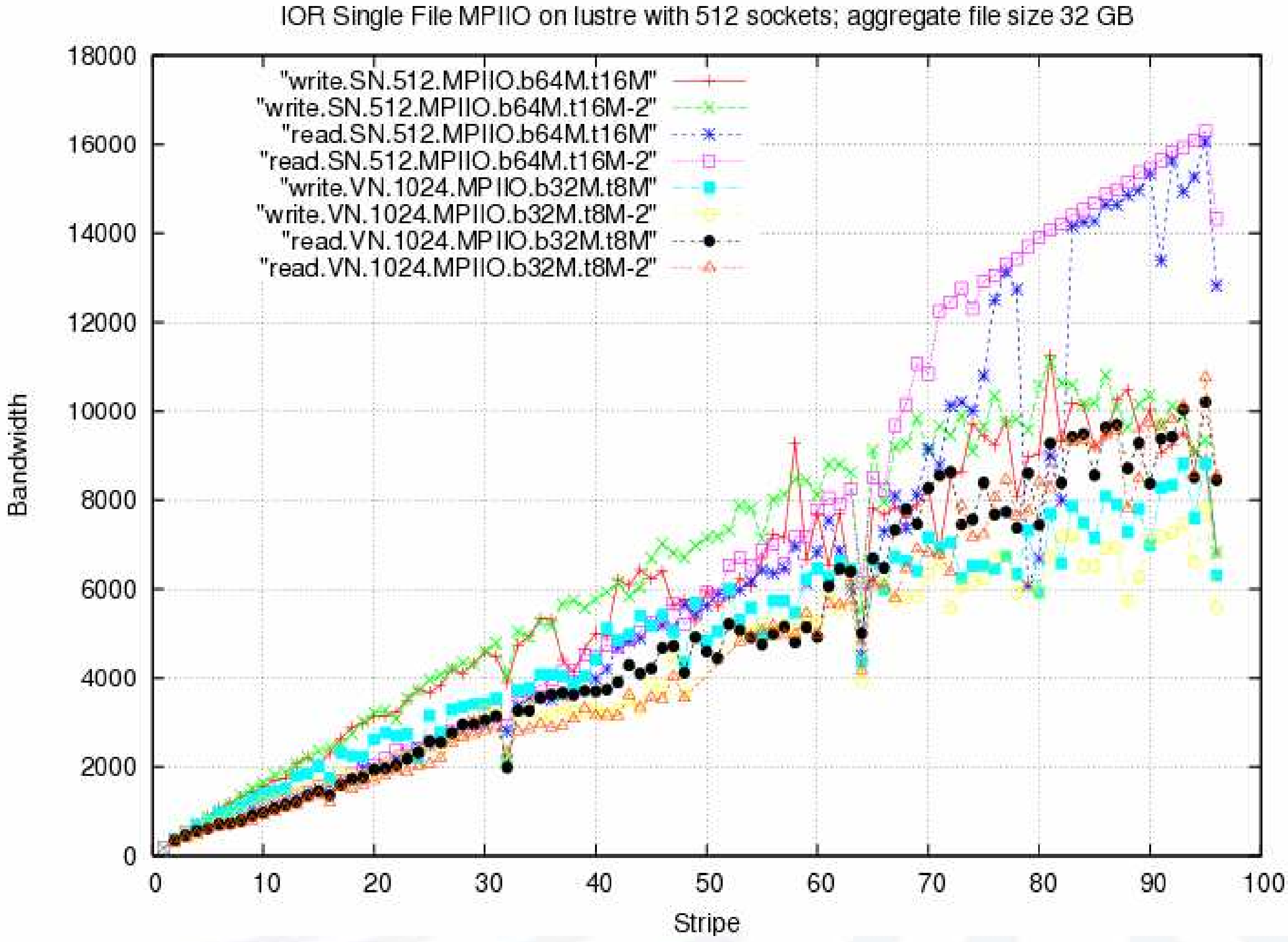

20 Writers, 150 Stripes


50 Writers, 150 Stripes

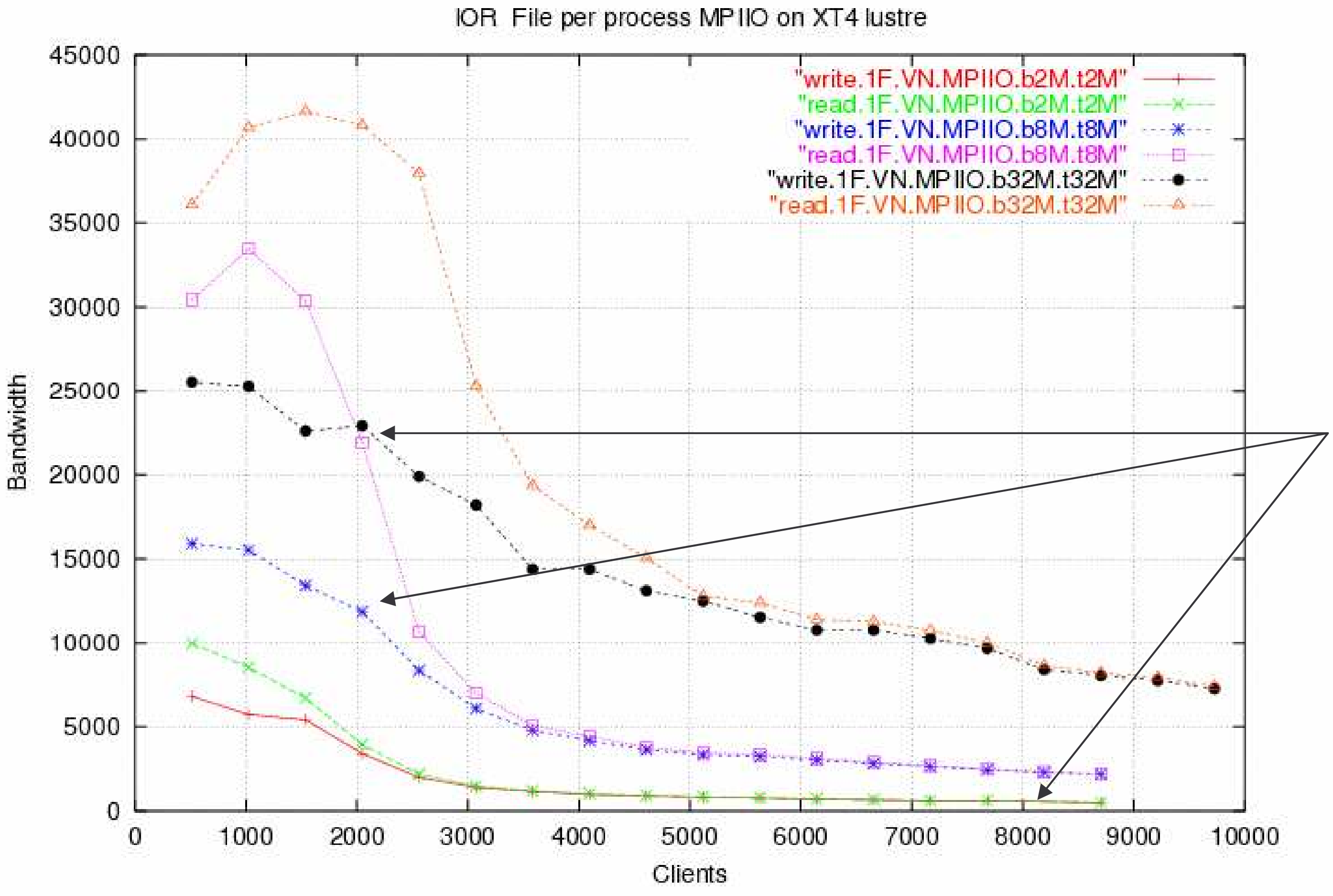

150 Writers, 150 Stripes


300 Writers, 150 Stripes

IOR: XT3 Striping, lustre 1.5.25, 96 OSTs

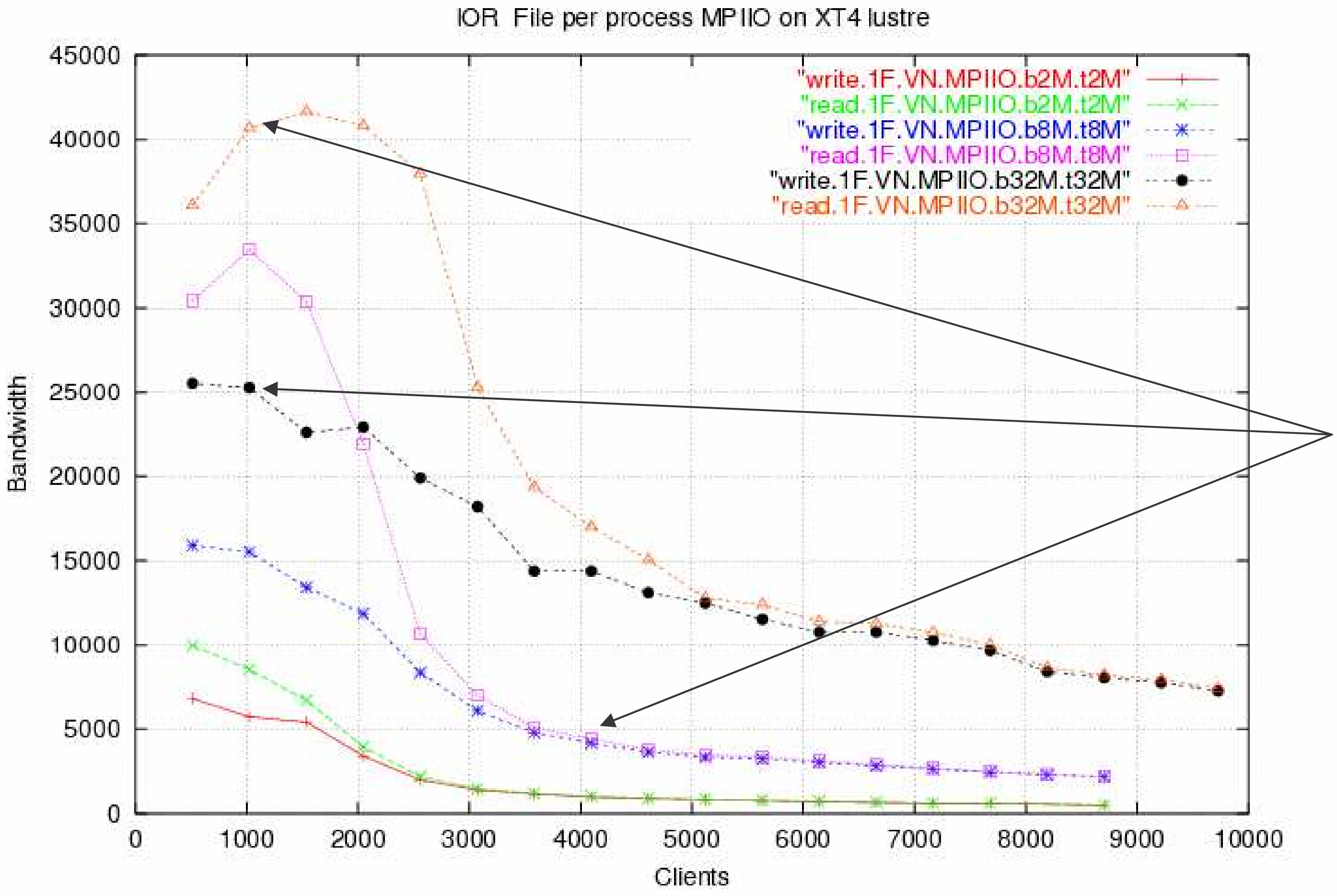


IOR: XT4, 1.5.31, pgi/6.2.5, 144 OSTs



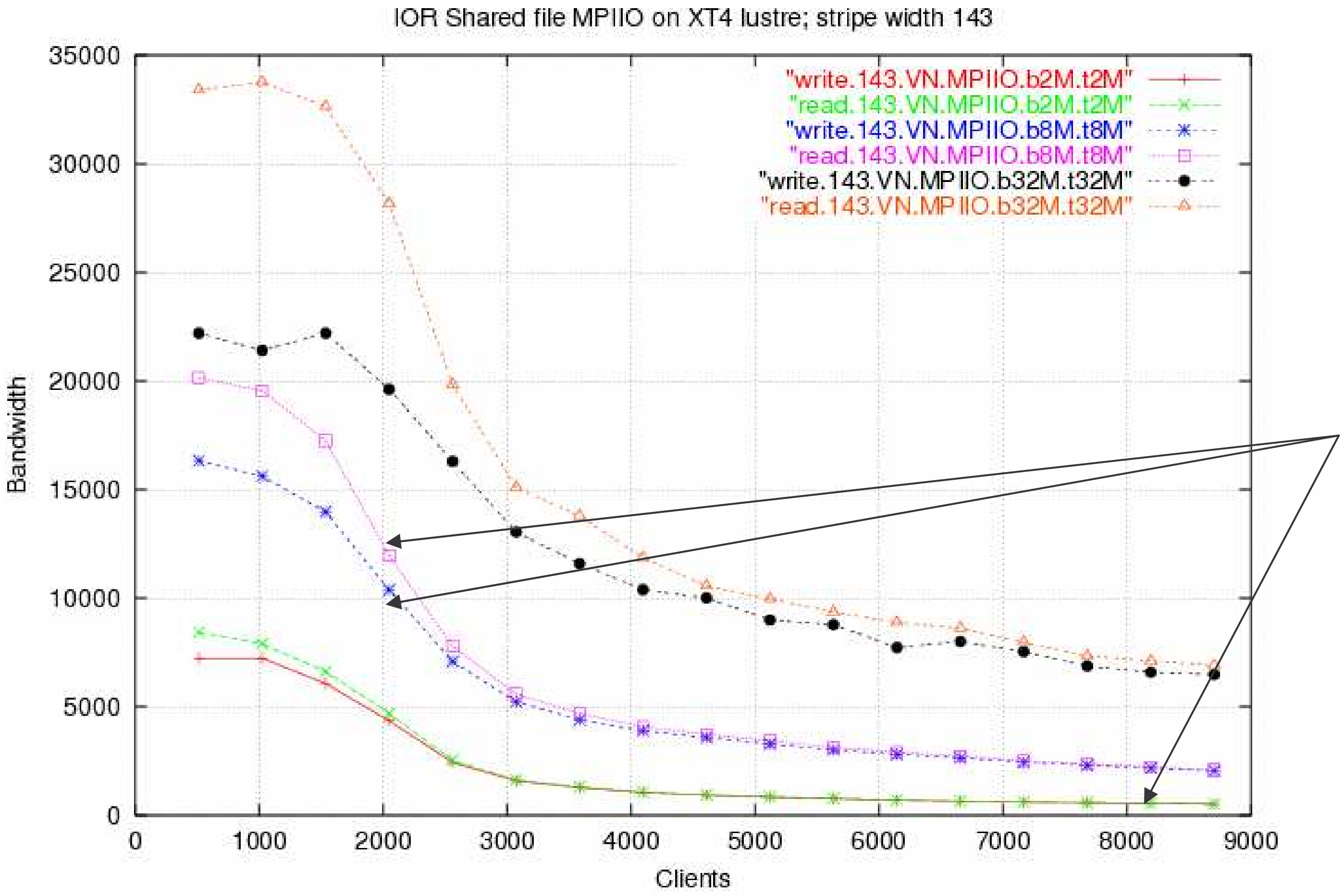
Total IO of 16 GB, but using fewer IO nodes better by 10x for writes and 20x for reads

IOR: XT4, 1.5.31, pgi/6.2.5, 144 OSTs



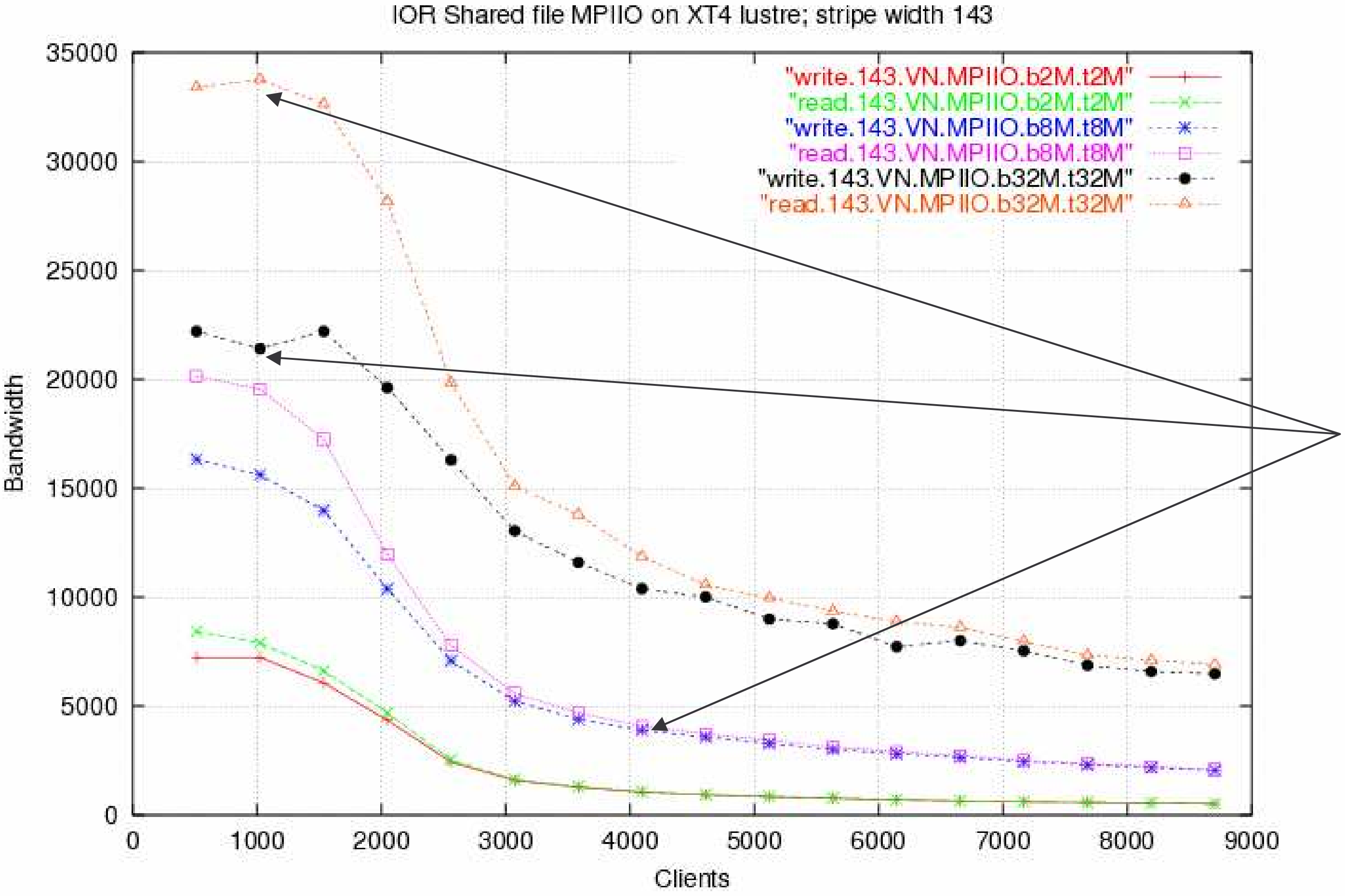
Total IO of 32 GB, but using fewer IO nodes better by 5x for writes and 8x for reads

IOR: XT4, 1.5.31, pgi/6.2.5, 144 OSTs



Total IO of 16 GB, but using fewer IO nodes better by 10x for writes and reads

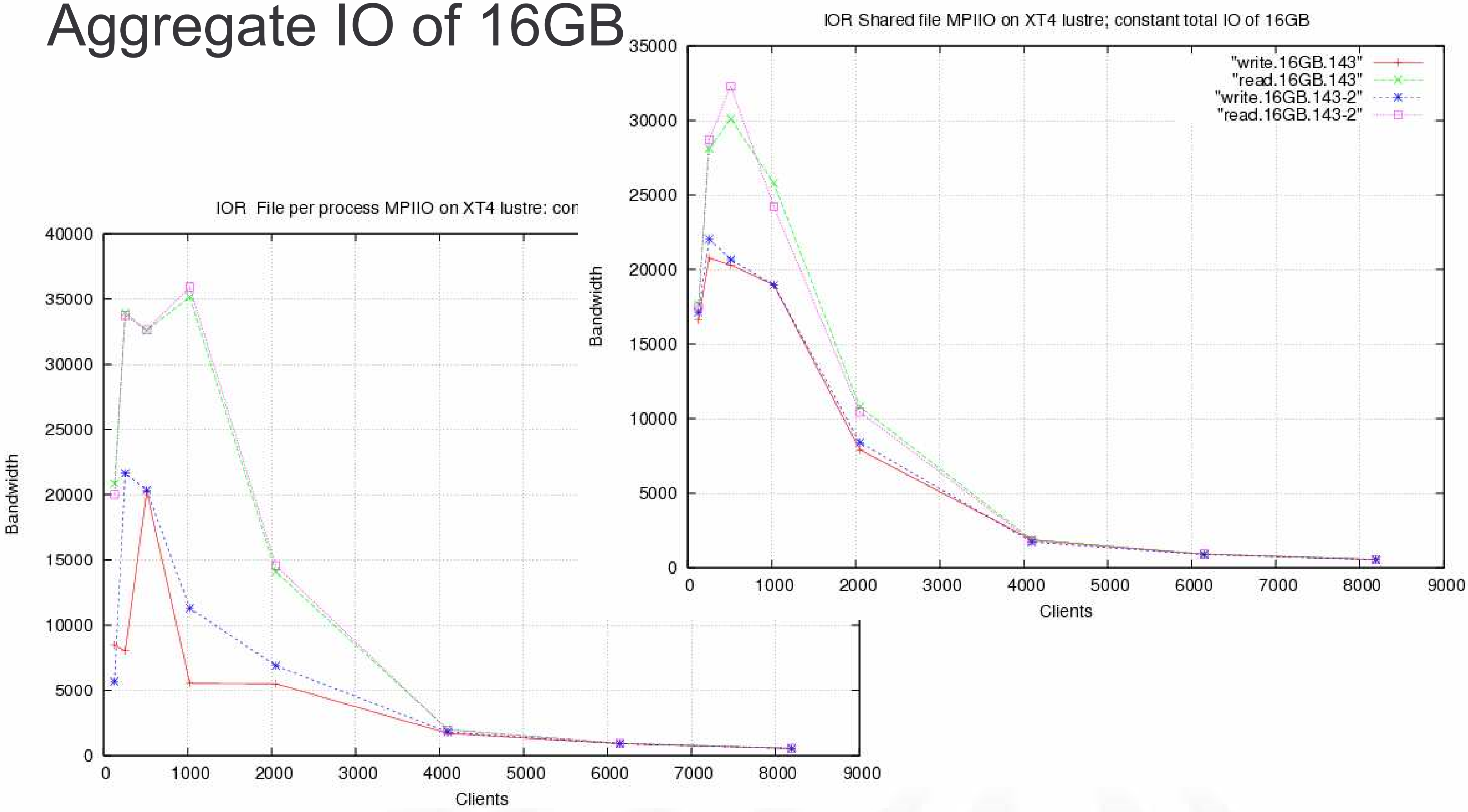
IOR: XT4, 1.5.31, pgi/6.2.5, 144 OSTs



Total IO of 32 GB, but using fewer IO nodes better by 5x for writes and 8x for reads

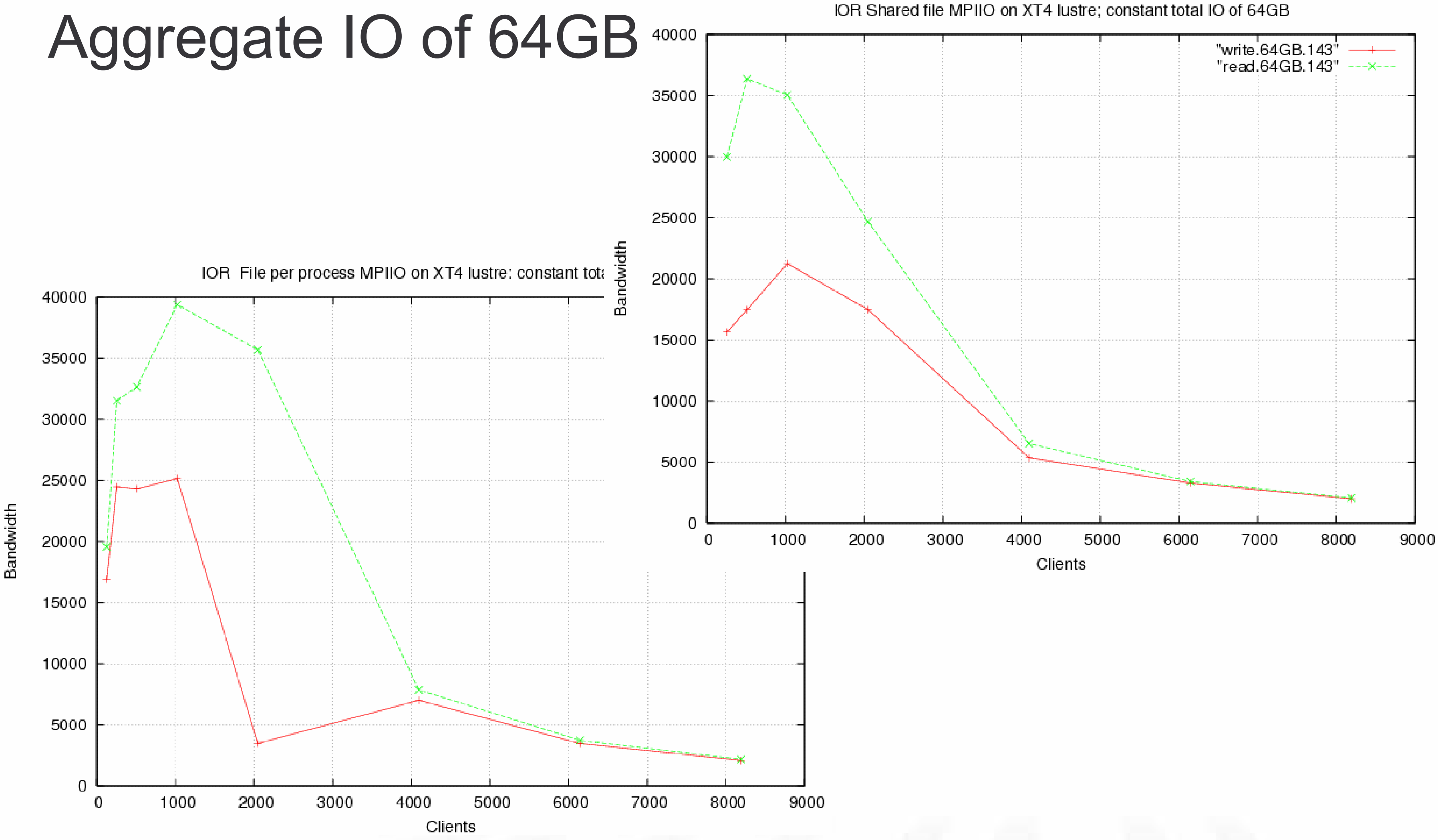
IOR: XT4, 1.5.31, 144 OSTs

Aggregate IO of 16GB



IOR: XT4, 1.5.31, 144 OSTs

Aggregate IO of 64GB



Suggested Guidelines

- **Do Large I/O Operations in Parallel with MPI-IO**
- **Create a natural partitioning of nodes so that data will go to disk in a way that makes sense**
- **Stripe as close to the maximum OSTs as possible given your partitioning**
- **Use buffers of at least 1MB, 10MB if you can afford it**
  - On XT, try **IOBUF** - I/O buffering layer (no MPI-I/O)
    - It works and requires no code changes
      - Can buffer stdout too
  - Future IOBUF or MPI I/O Hints may help MPI-I/O
- **Make your I/O flexible so that you can tune to the problem and machine**
  - One hard-coded solution will meet your needs some of the time, but not all of the time
  - Use a subset of nodes for I/O (make this tunable) when running large-scale
    - According to recent tests, 4-8 x the number of OSTs
  - MPI I/O hints would be a portable solution (still verifying XT)
- **We're looking into whether the buffering and subsetting can be easily handled with "hints"**

Suggested Guidelines (cont.)

- **On parallel HDF5 and parallel-netCDF**
  - General consensus at Cray Technical Workshop is that they perform very poorly, lustre or not.
    - I know this is not what you want to hear
  - We hear that people are working on it
- **Everyone opening a file at the same time at scale is sure to be slow, offset if possible**
- **Performance will be variable**
  - Lustre filesystem is a shared resource

Future Work and Acknowledgements

- Future Work
  - MPI I/O Hints
  - Verify Results with Libraries (HDF5, NetCDF, pNetCDF)
  - Real Application Results
- Acknowledgements
  - This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
  - Benchmark 1 data courtesy of Gene Wagenbreth
- Questions/Comments?
- We’ll be hosting a BOF at 5pm on Wednesday for further discussions.