XT Parallel IO

NCCS USERS MEETING

Mark Fahey
faheymr@ornl.gov
March 29, 2007

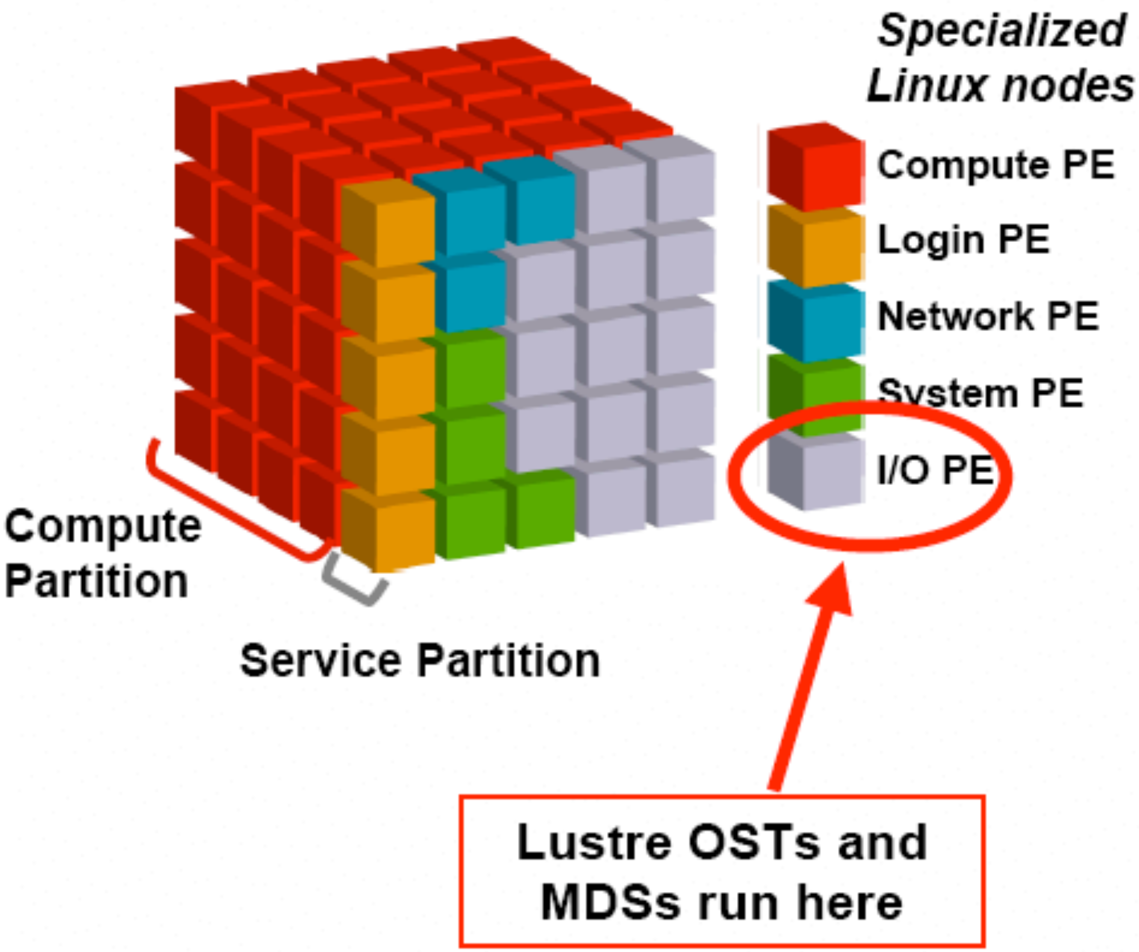
Outline

- Jaguar Lustre overview
  - System architecture
  - Lustre Terminology
  - Commands
  - Limitations
- Brief Endian-ness discussion
- Parallel I/O at scale
  - Basic parallel I/O methods
  - Problem with typical methods
  - A solution
  - Benchmarks
- Research

Jaguar Lustre overview

- System architecture
- Lustre Terminology
- Commands
- Limitations

Jaguar XT3/4 Architecture



- Compute partition has
  - 11,508 AMD dual-core processors
  - 46 TB of memory
- Lustre filesystems
  - Serviced by 80 I/O nodes
  - /lustre/scr144
    - 144 OSTs
    - Peak is 72 GB/s
    - Practical ~48 GB/s
    - Early results
      - Read 45 GB/s
      - Write 25 GB/s
  - /lustre/scr72[a,b]
    - 72 OSTs each
    - Default scratch

Lustre terminology

- The concept of object storage is basic to Lustre
  - Objects can be thought of as inodes and are used to store file data. Lustre inodes simply contain references to the object storage target (OST) that stores the file data
  - Access to these objects occurs through object storage servers (OSSs), which provide the file I/O service
  - The OSTs perform the block allocation for data objects, which results in distributed and scalable allocation

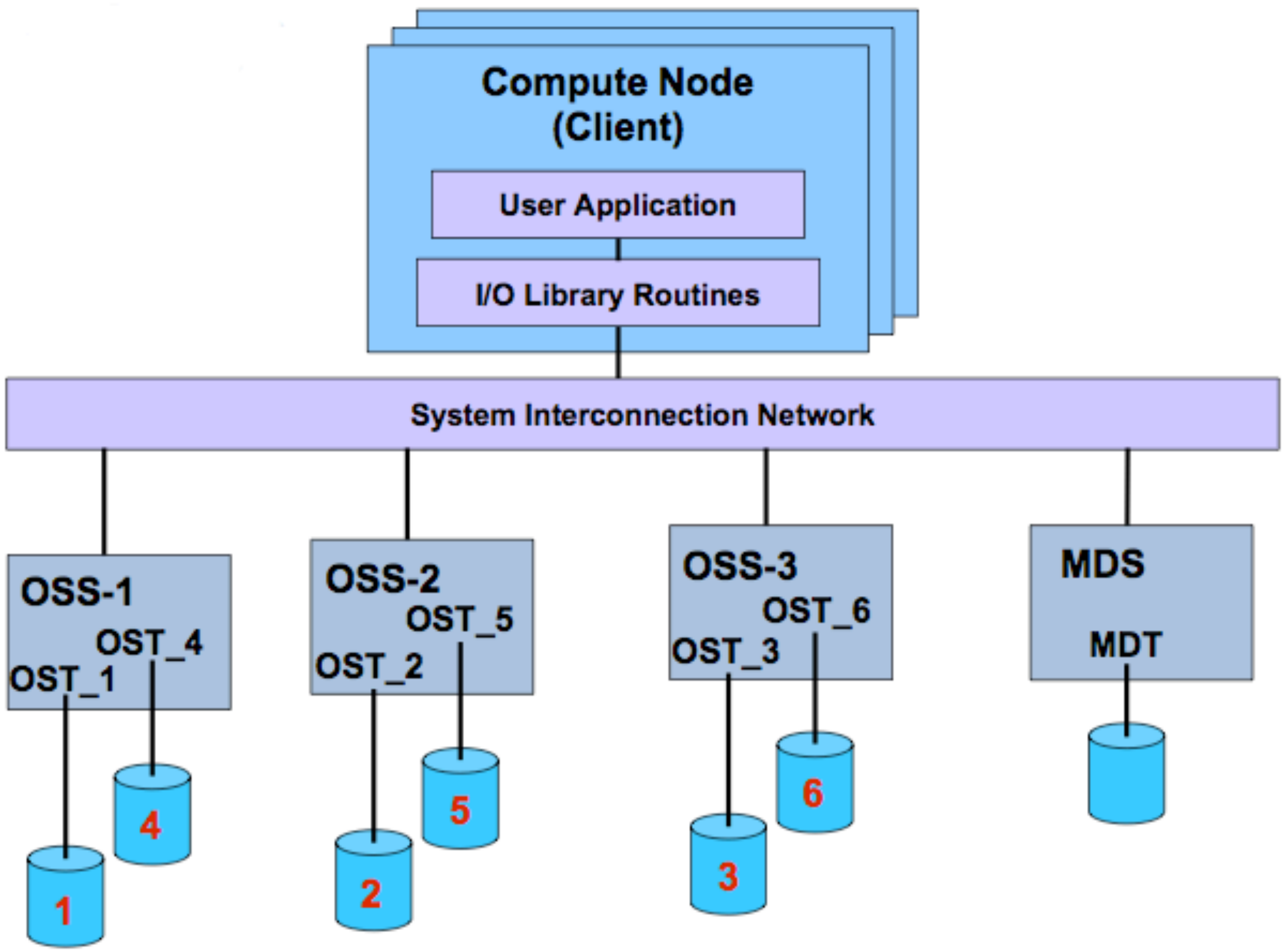
Lustre terminology (cont.)

- The namespace is managed by metadata services that manage the Lustre inodes
  - The services perform file lookups, file creation, file and directory attribute manipulation
  - Such inodes can be directories, symbolic links, or special devices
  - The associated data and metadata is stored on the metadata servers

Lustre terminology (cont.)

- MDS - metadata server
  - The Server node
- MDT - metadata target
  - This is the software interface to the backend volume
  - Controls filesystem metadata (inodes) and locking mechanism
    - The backend volume is an ext3 file system
    - LUNs are formatted with 4096 byte blocks
- OSS - object storage server
  - The server node
  - Support multiple OSTs
- OST - object storage target
  - This is the software interface to the backend volume
    - The backend volume is an ext3 file system
    - LUNs are formatted with 4096 byte blocks
    - The multi-block allocator (MBA) (Linux 2.6) is used for performance
    - LUN size is limited to 2 TB

XT3/4 Lustre Architecture



- XT compute clients run Catamount microkernel and use liblustre
- Portals networking over XT SeaStar interconnect
- Full Linux on service and I/O nodes

Lustre commands

- lfs - Lustre utility that can be used to create a file with a specific striping pattern, displays file striping patterns, and find file locations
  - Suboptions: setstripe, getstripe, find, help
- Examples
  - set stripe width (count) to 1 on <dir>
    - `lfs setstripe <dir> 0 -1 1`
  - find stripe width on <filename> with minimal output
    - `lfs find --quiet <filename>`
  - find stripe width on <filename> with default output
    - `lfs getstripe <filename>`
  - Set stripe size (per striped OST) to 2MB on <dir>
    - `lfs setstripe <dir> 2097152 -1 1`
  - Get online help
    - `lfs help <suboption>`

Lustre limits

- The maximum file size is 320 TB (on any lustre)
  - Maximum number of stripes per file is 160
    - 2 TB x 160 = 320 TB (2 TB is max LUN size)
- Limits on Jaguar

Limits	Max Stripe count	Capacity
/lustre/scr144	144	288 TB
/lustre/scr72[a,b]	72	144 TB

  - scr72a and scr72b don’t overlap
  - scr72[a,b] overlaps half of scr144

Endian-ness

- Little-endian
  - x86 machines (Intel, AMD), DEC Alpha
    - XT3/4
- Big endian
  - X1[E], IBM PPC (including BG/L), MIPS, Sparc
- Many compilers provide bi-endianness support for Fortran binary files (Intel, PGI, etc)
  - But is there a price to pay?

Endian-ness

- One can use the PGI `-byteswapio` option to swap the endian-ness for Fortran I/O
  - **You can use this on a subroutine by subroutine basis**
- Creating a 4GB file (512 8MB writes) with one process, sequential unformatted (on XT3)
  - Default: ~18 MB/S
  - byteswapio: ~9.4 MB/s
- User called endian swap
  - Can get ~15 MB/s
  - ** But control words will be different endian-ness than the data
- Very similar results using direct unformatted (simulating sequential I/O)

Endian-ness discussion

- So there is a cost (50%) when doing Fortran unformatted I/O with one process
  - Can this cost be amortized away in parallel?
  - With 100 or more processes, the cost is reduced to ~23% hit
    - 96 processes in SN mode writing 16 MB each
      - default 7.1 GB/s
      - byteswapio 5.4 GB/s
    - 192 processes in VN mode writing 16 MB each
      - default 11.7 GB/s
      - byteswapio 9.2 GB/s
- This may matter if the I/O cost in your code is significant

Parallel I/O at scale

- Basic parallel I/O methods
  - Problem with typical methods
  - A solution
- Benchmarks
  - Striping
  - Buffer sizes
  - Subsetting

Parallel I/O in general

- Two common methods:
  - All data is reduced to 1 process which does I/O
  - All tasks do I/O
    - A file read/written by each task
      - Independent files
    - All tasks read/write a part of one file
      - shared file
- With both methods, typically one uses
  - Fortran or C I/O with MPI
    - Records/seeks for shared file
  - MPI I/O
  - Parallel HDF5 or netCDF
    - won't be talking about these today

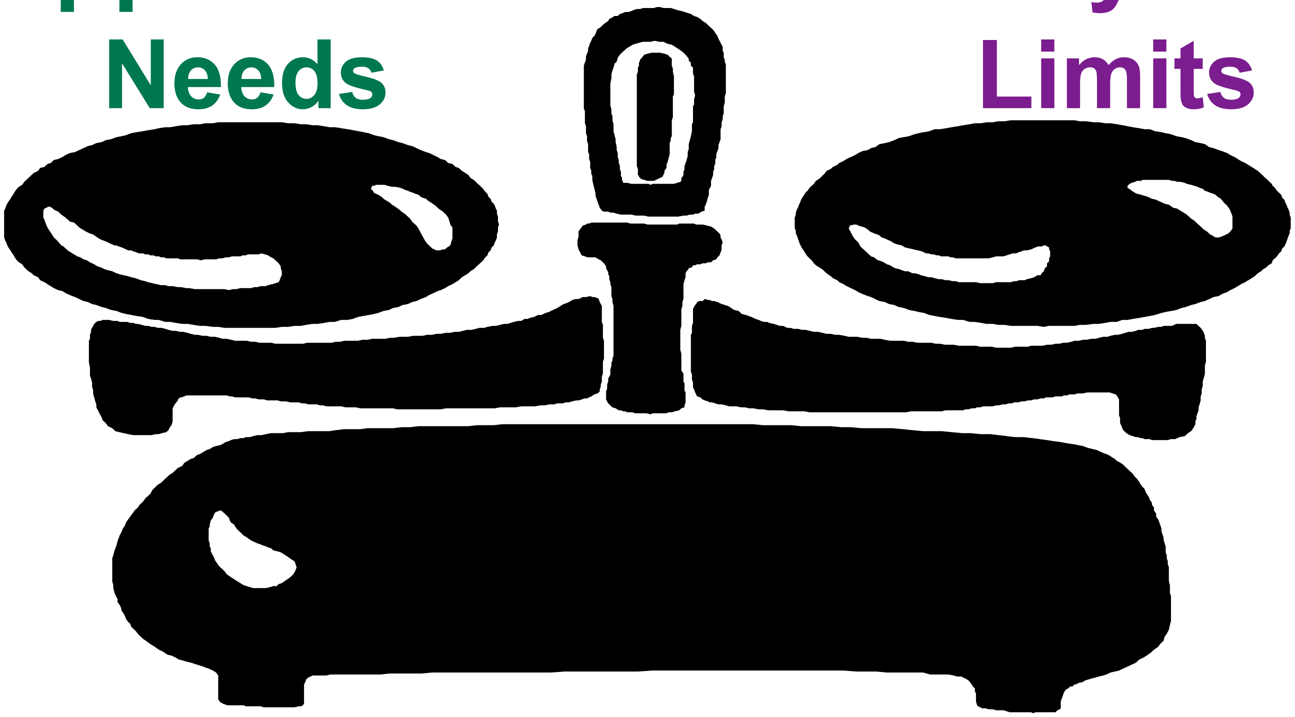
The Problem

- 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 (200 MB/s)
  - All processes read/write at the same time
    - Slow opens (all hit the MDS at the same time)
    - Overwhelm OSTs and/or IO service nodes
    - Possibly inconvenient to users

Striking a Balance

Application Needs

Filesystem Limits



Subset of readers/writers

- 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
  - I/O Buffering
  - High Bandwidth, Low FS Stress
- The Costs
  - I/O Nodes must sacrifice memory for buffer
  - Requires Code Changes

Subset of readers/writers (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
  - there are approximately the same number of groups as OSTs
- 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/write shared disk file

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)
```

Benchmarks

- Topics discussed
  - Lustre striping
  - Buffer sizes
  - Subsetting

Caveats

- OS level 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

Striping

- Lustre has the flexibility to specify how a file is striped across OSTs
  - Default set when file system is made
  - User can specify with lfs setstripe [dir | file] ...
- Striping across multiple OSTs is useful when an application writes large, contiguous chunks of data
  - OSTs run in parallel, increasing I/O performance
- If the application isn't writing large data, striping will hurt
  - Don't stripe for small files

Benchmark Results: 1 I/O Node - Stripes

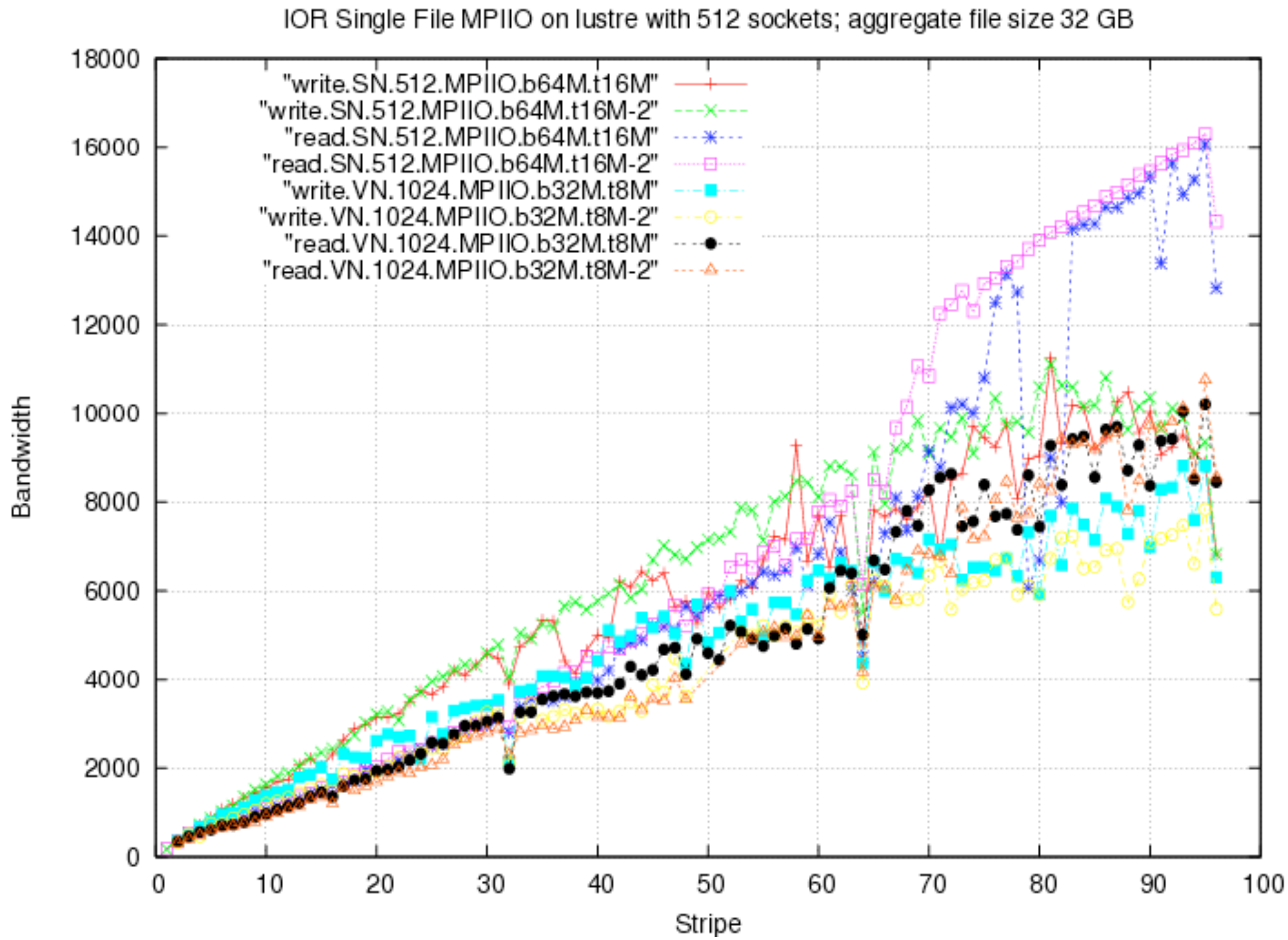
- Single IO node, 10 megabyte buffer, 20 megabyte stripe size: bandwidth of IO write to disk

Number of stripes

1	10	50	100	150	160
150MB/s	134MB/s	135MB/s	139MB/s	149MB/s	148MB/s

- Using a single IO node:
  - number of stripes doesn't matter
  - stripe size doesn't matter (timings not shown)

XT3 Striping, lustre 1.5.25, 96 OSTs



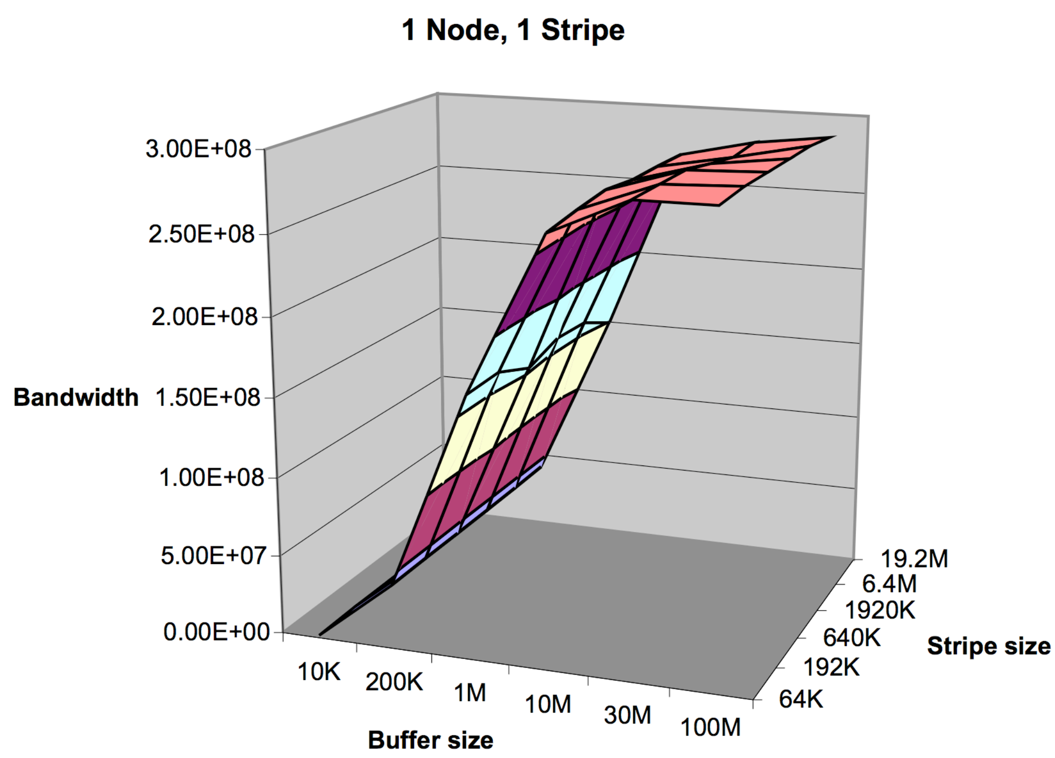
Striping discussion

- From the data, we see
  - Don’t use multiples of 32
  - Don’t use max
    - Not sure if this applies to lustre config on XT4?

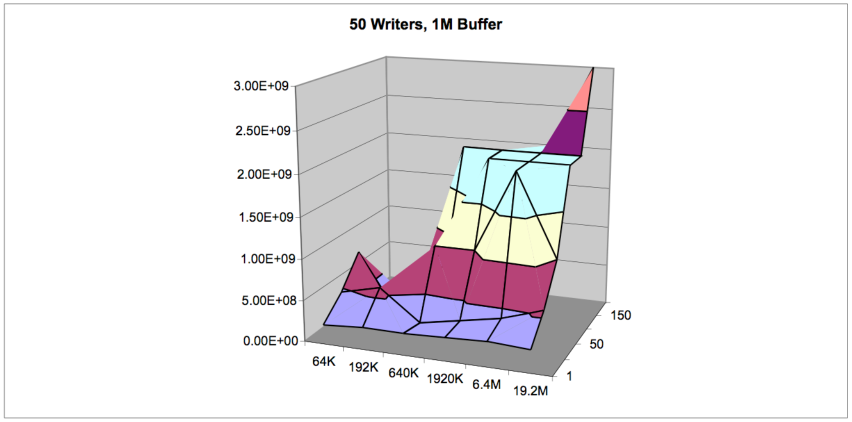
Buffer sizes

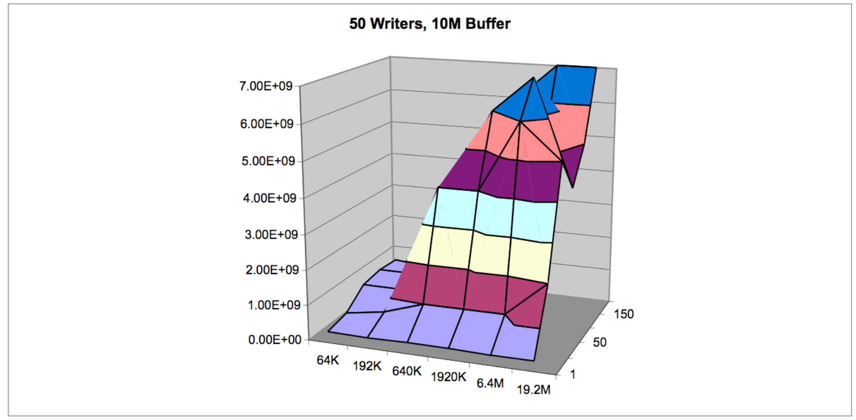
Benchmark Results: 1 I/O Node - Buffer Size

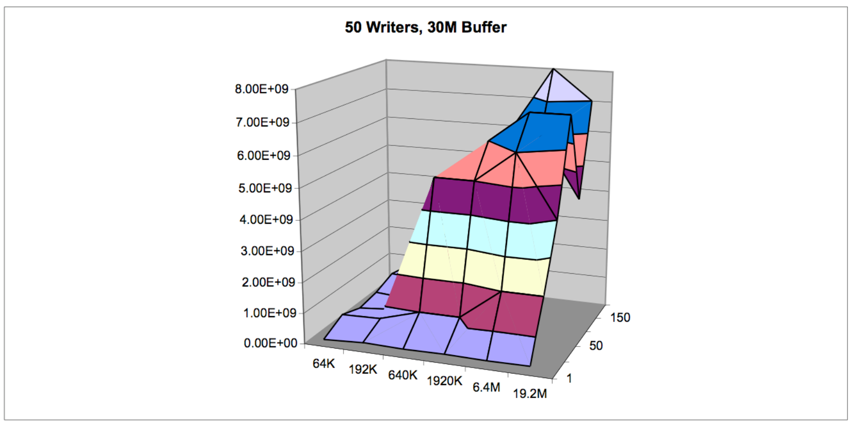
- Single node, single stripe: bandwidth of IO write to disk for different buffer sizes
  - Buffer size is the size of contiguous memory on one IO node written to disk with one write
- Buffer size should be at least 10 megabytes

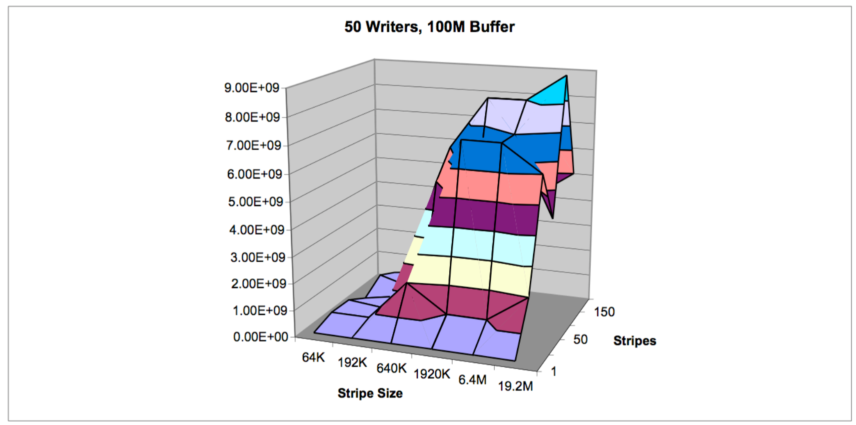


50 Writers, Varying Stripe Count, Size and Buffer Size

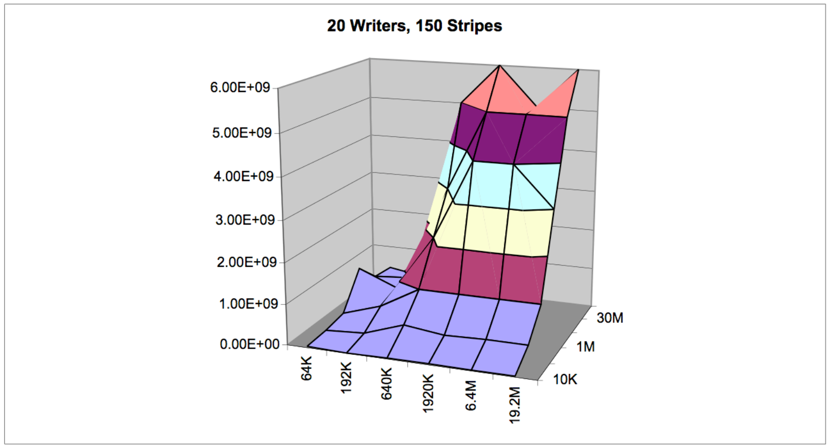


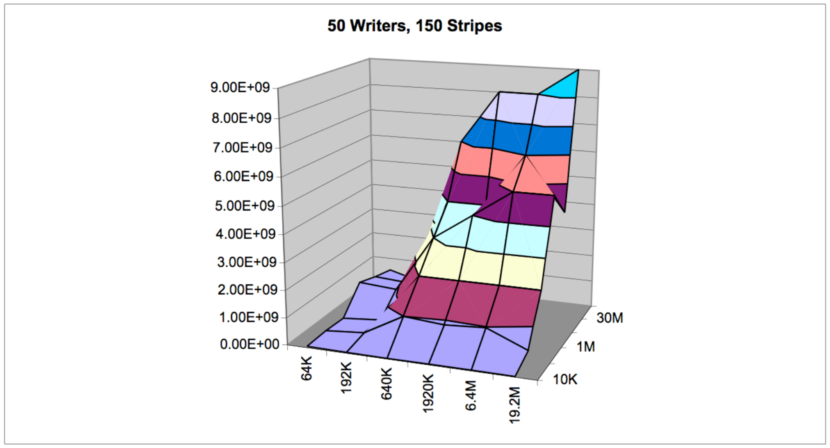


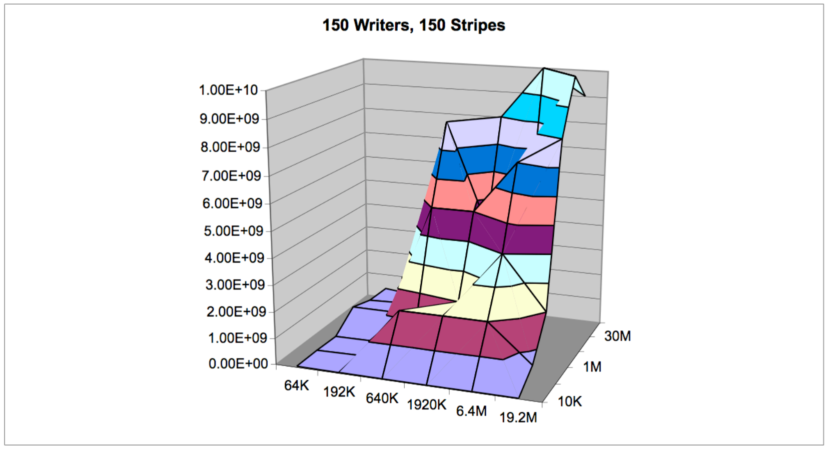


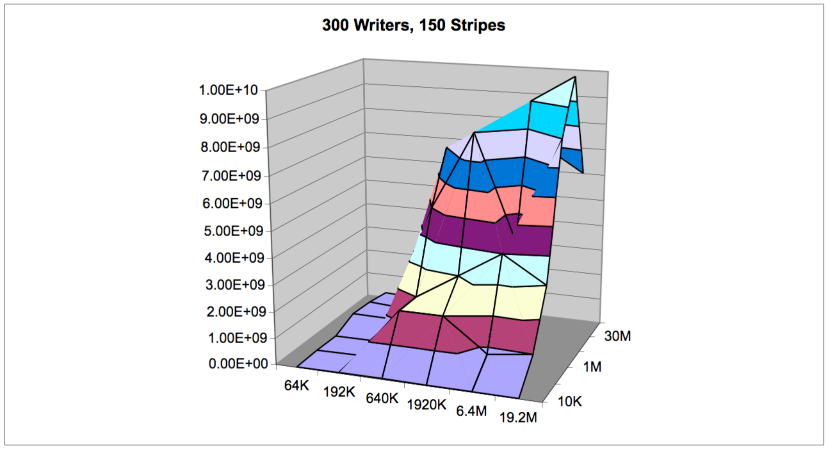


150 Stripes, Varying Writers, Buffer, and Stripe Sizes







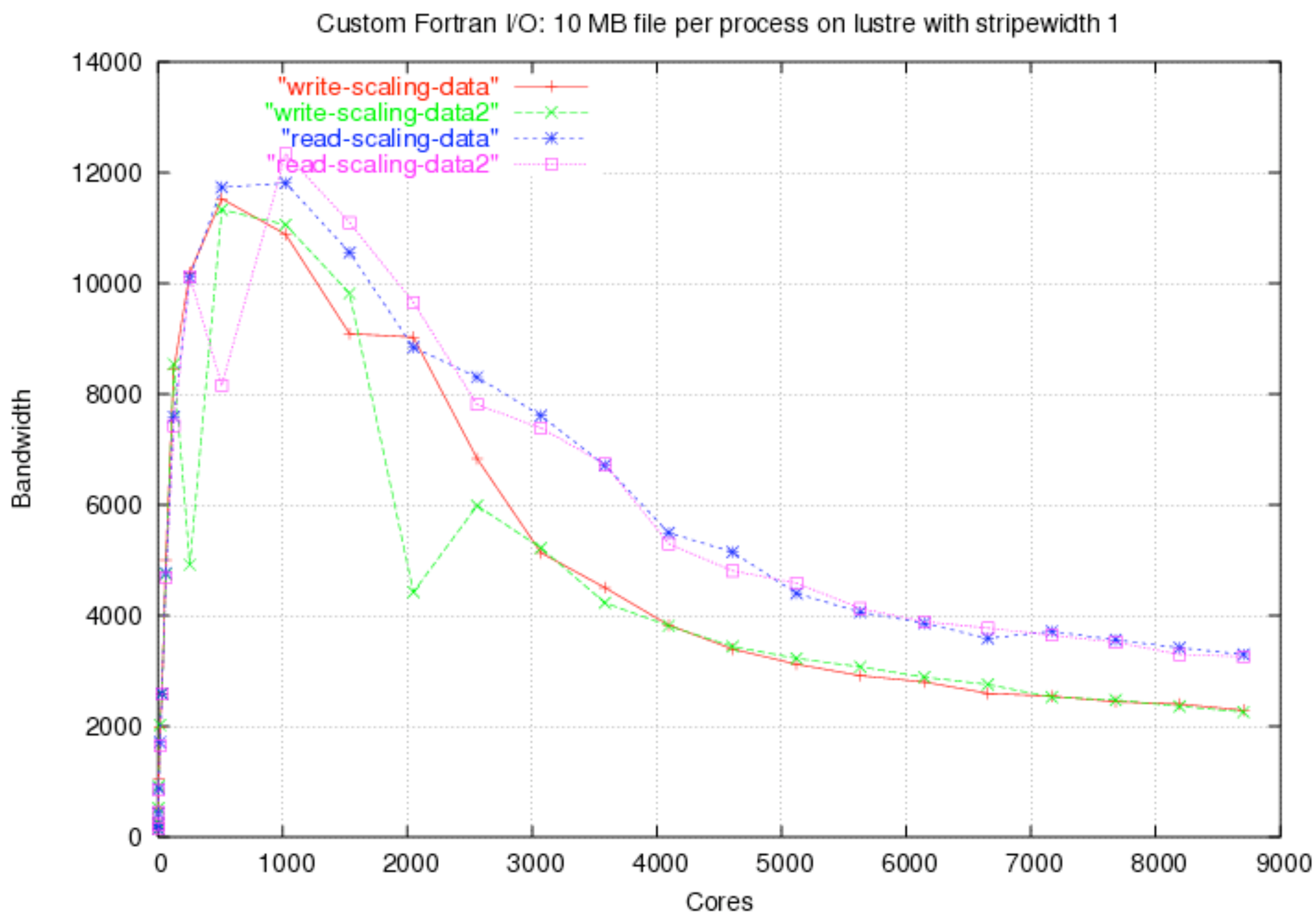


Scaling clients

- Will now show benchmark data of scaling the number of IO clients, with
  - Custom MPI/Fortran code
  - IOR

Parallel Fortran I/O

- 10 MB file per process
- stripewidth of 1
- XT3
  - 1.5.29
  - pgi/6.1.4
  - 96 OSTs

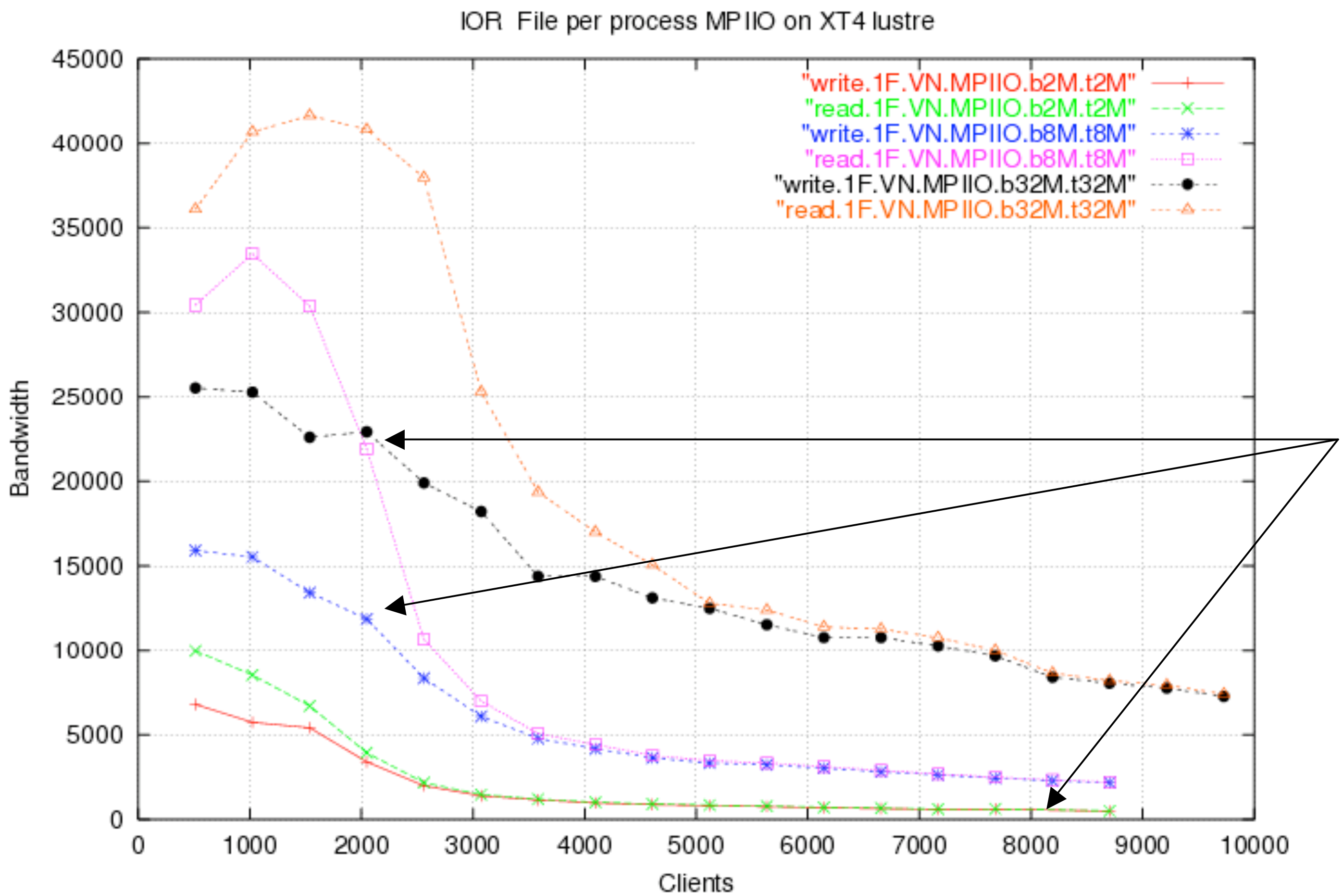


Parallel Fortran I/O (cont.)

- This plot tells us ….
  - Sweet spot around 512-1024 writers
  - At full size
    - 2 GB/s writes, 3 GB/s reads
  - Reads faster than writes >= 1024 writers

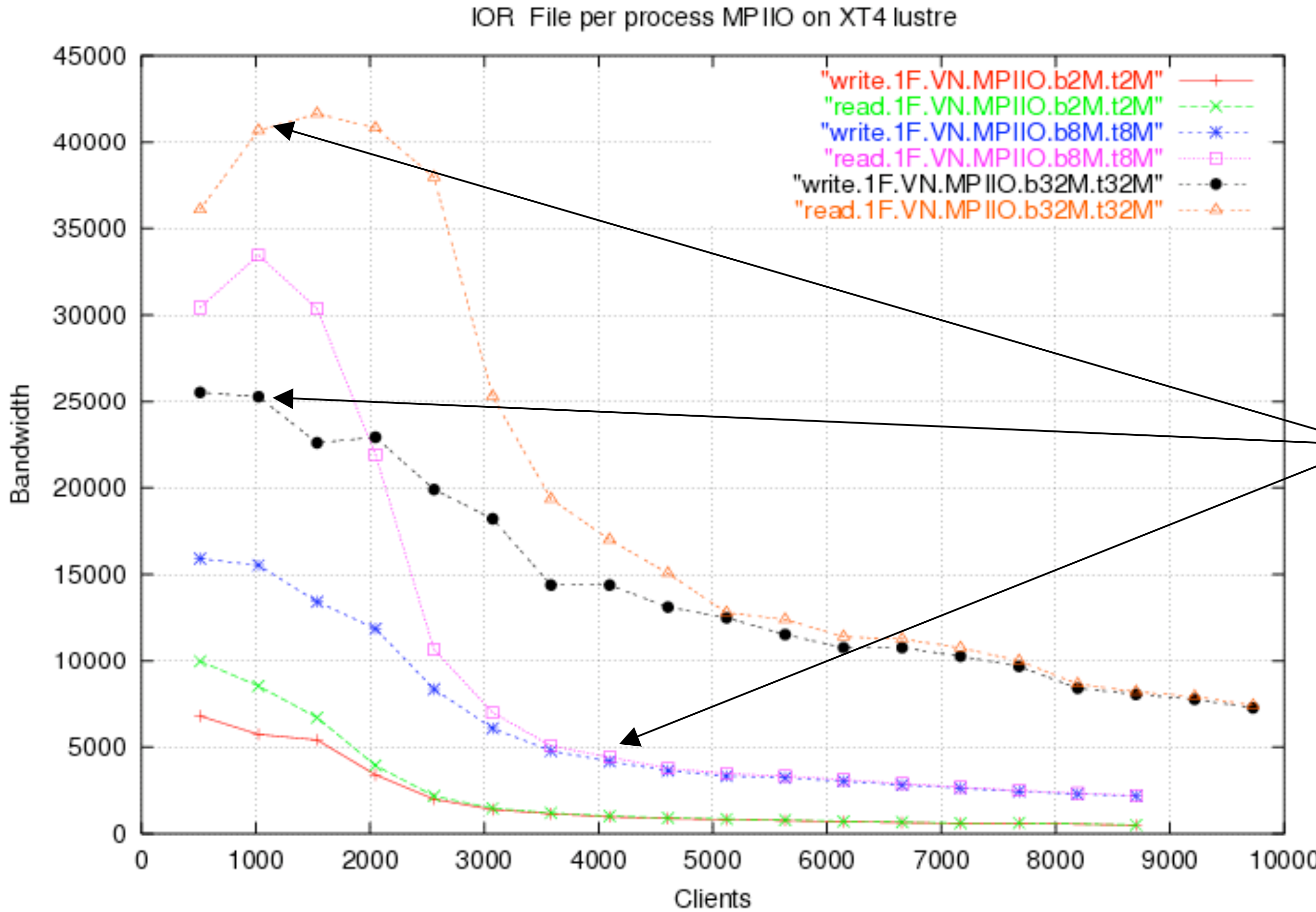
* Data taken in non-dedicated mode

XT4, 1.5.31, pgi/6.2.5, 144 OSTs



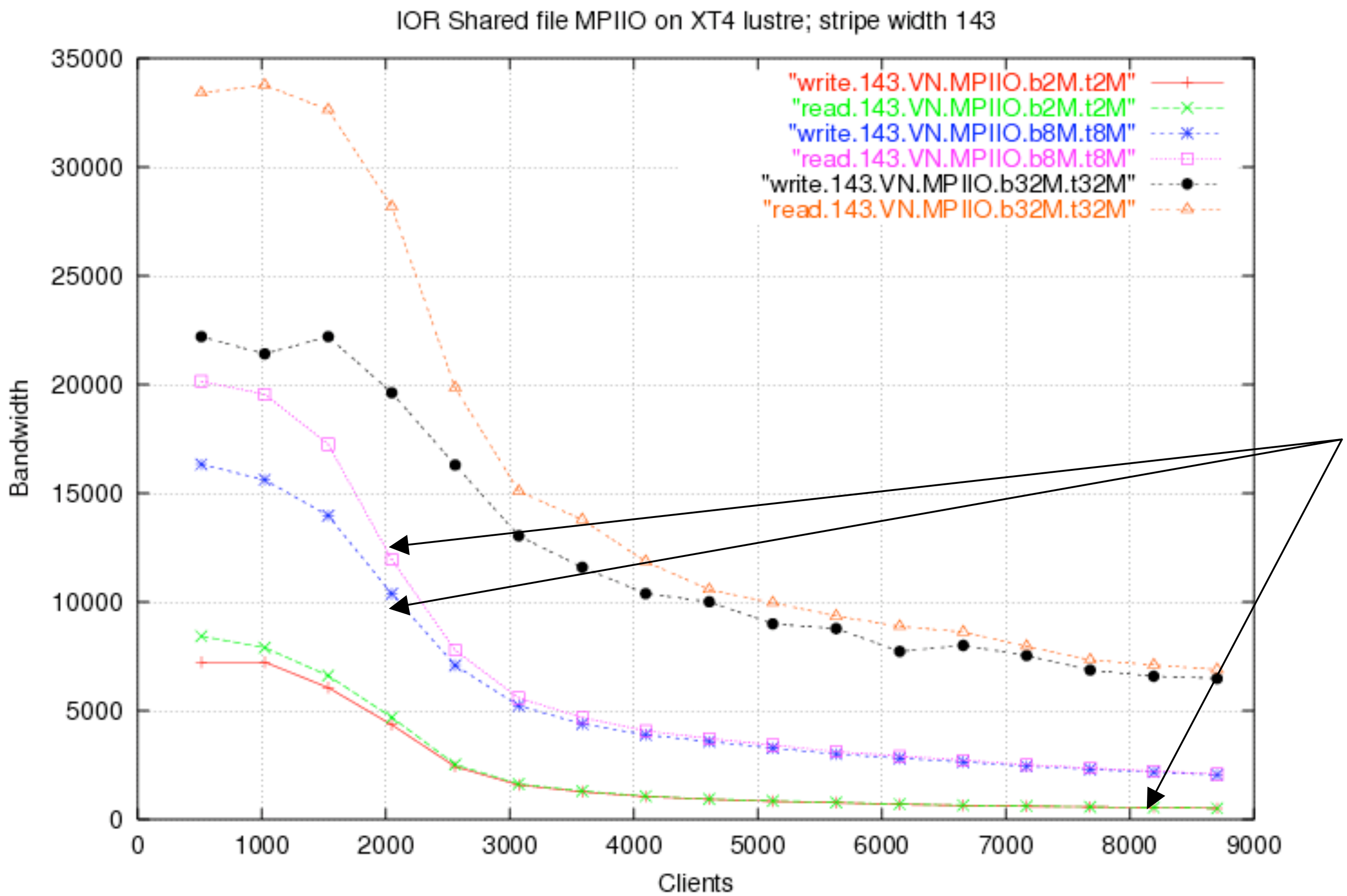
Total IO of 16 GB, but using fewer IO nodes better by 10x for writes and 20x for reads

XT4, 1.5.31, pgi/6.2.5, 144 OSTs



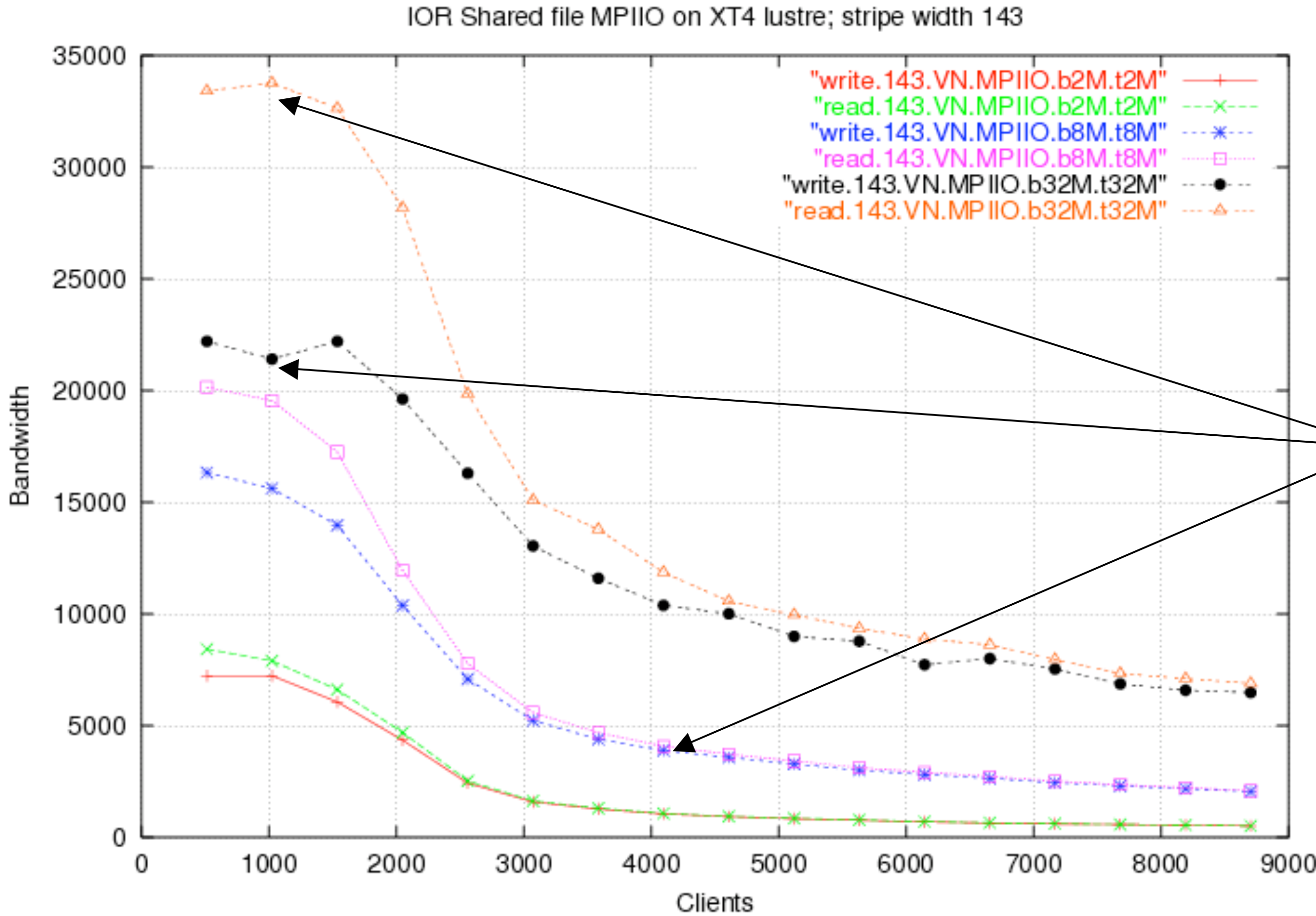
Total IO of 32 GB, but using fewer IO nodes better by 5x for writes and 8x for reads

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

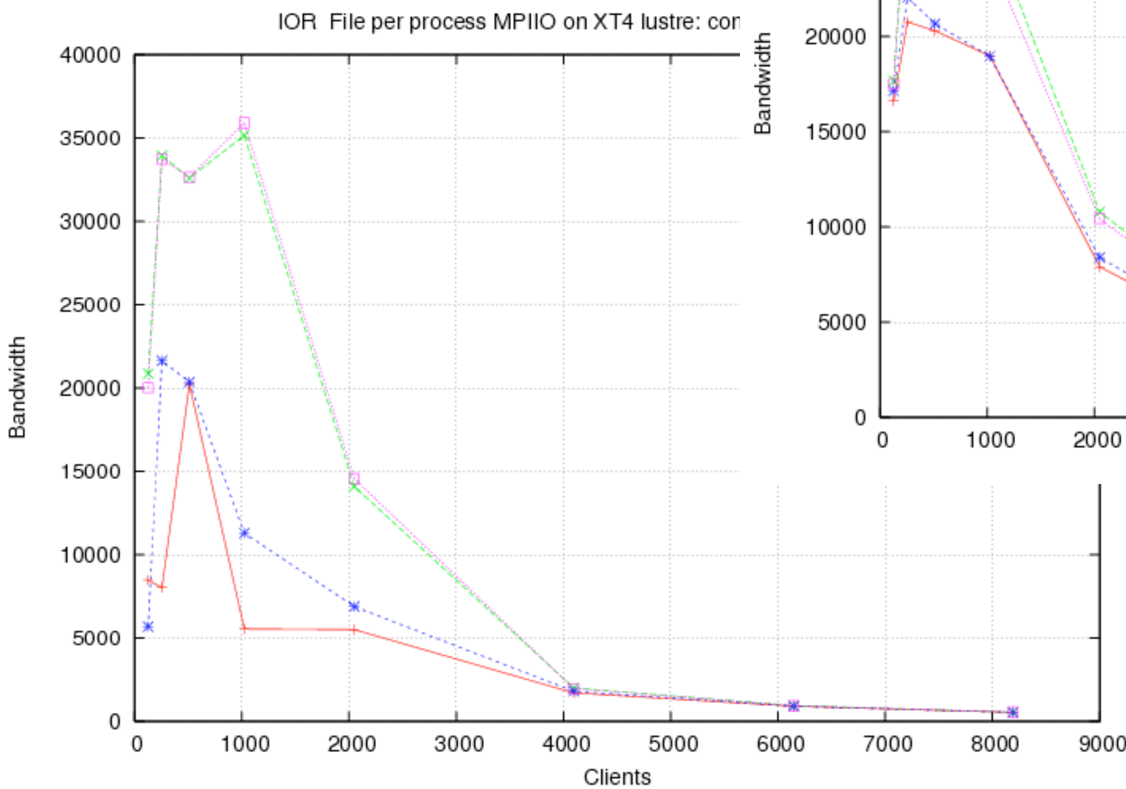
XT4, 1.5.31, pgi/6.2.5, 144 OSTs

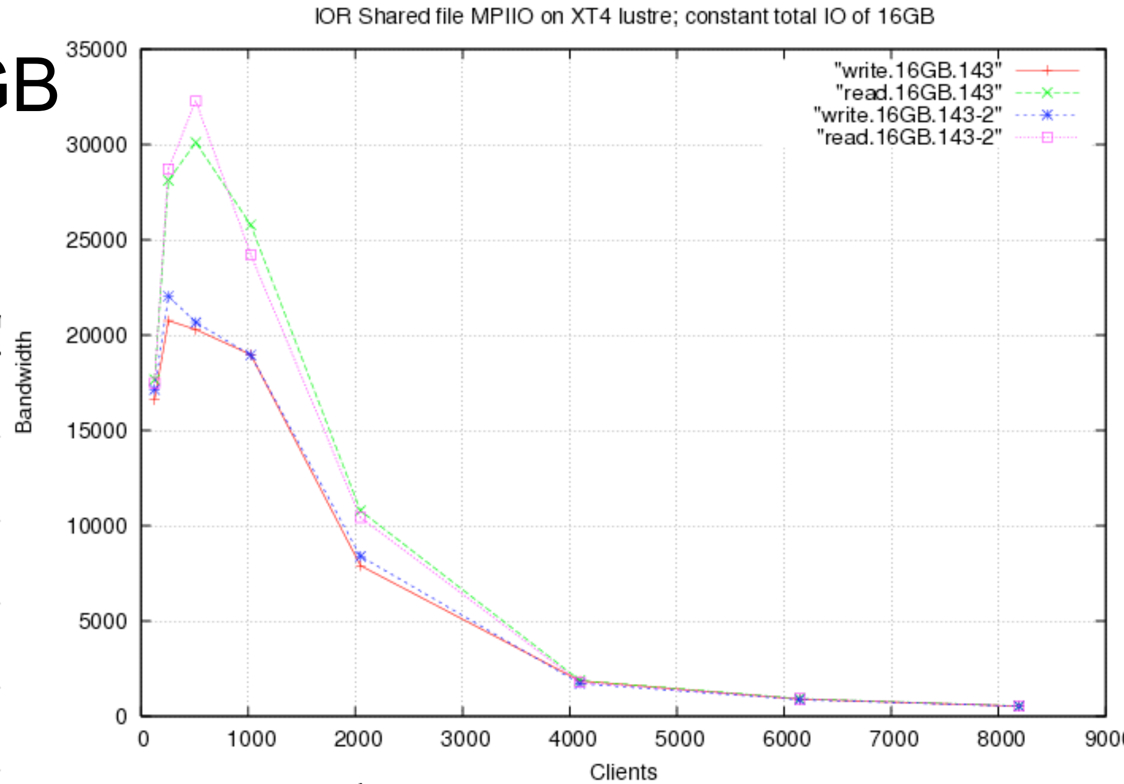


Total IO of 32 GB, but using fewer IO nodes better by 5x for writes and 8x for reads

XT4, 1.5.31, 144 OSTs

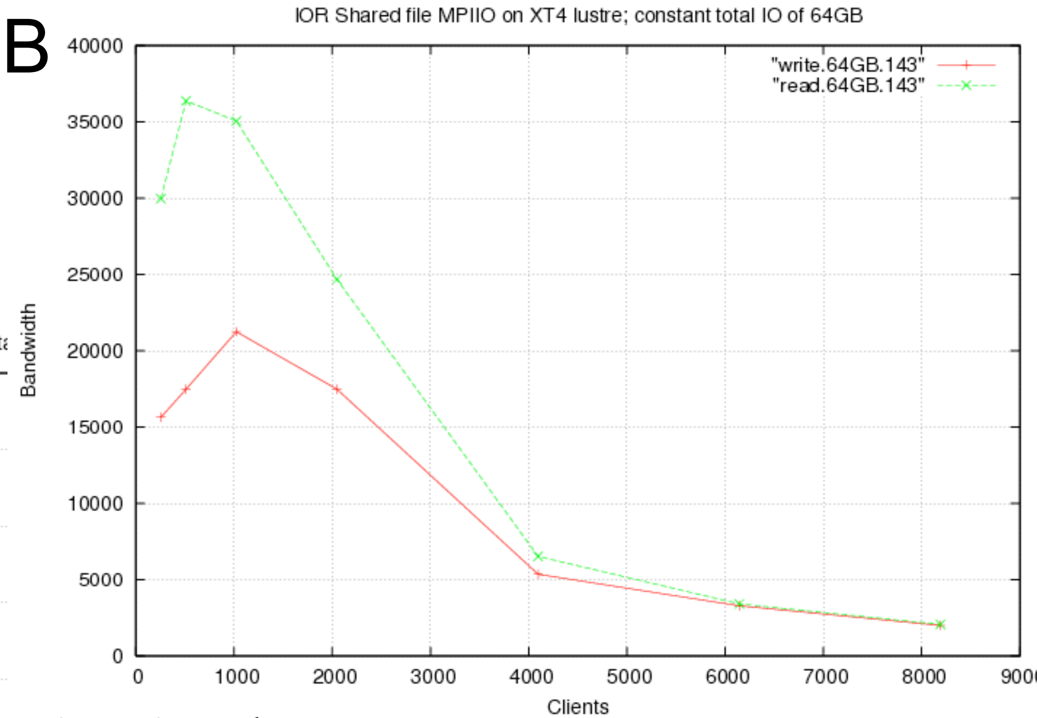
Aggregate IO of 16GB

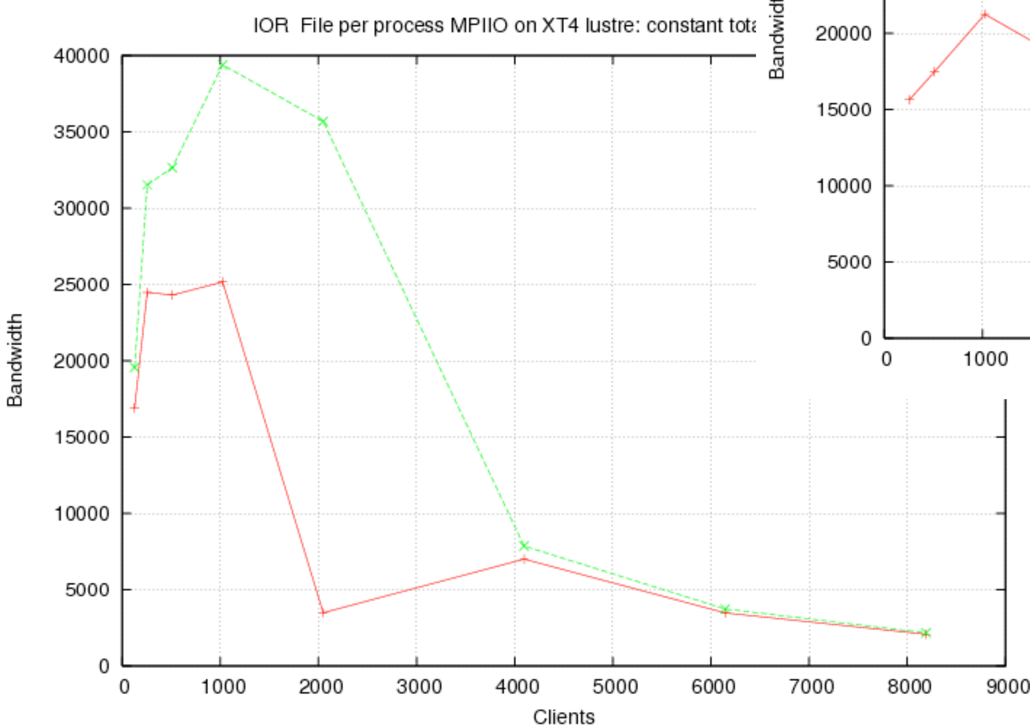




XT4, 1.5.31, 144 OSTs

Aggregate IO of 64GB





IOR scaling results

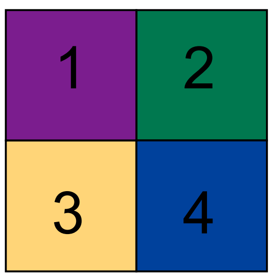
- These plots tell us ….
  - Larger IO buffers are better
  - Using fewer IO nodes at large scale is better
    - Optimal # of IO nodes is dependent on IO buffer size, data suggests
      - 2-8 x (# of OSTs) for 16GB aggregate file IO
      - 4-8 x (# of OSTs) for 64GB aggregate file IO

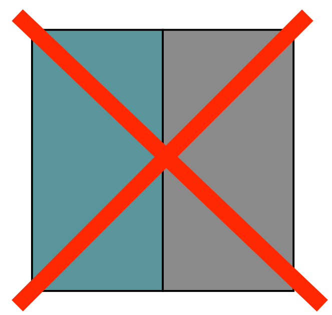
Writer/Reader Subsetting

- On ORNL's XT3 and XT4, sufficient evidence to conclude that too many readers/writers degrades IO bandwidth
  - Since the optimal number of IO nodes looks to be somewhere around 1024, we believe that using a subset of clients for IO is beneficial
- Goal: use subset of MPI processes to do IO
  - Shown to be more effective in previous slides
  - Aggregates IO too
  - Can't MPI IO does this automatically with hints?
    - Still investigating on XT
- Note: I have seen one plot (in a lustre tutorial class) of data from Sandia's Red Storm that shows almost no degradation from 2K clients out to 10K clients (40GB/s)
  - Unable to repeat this

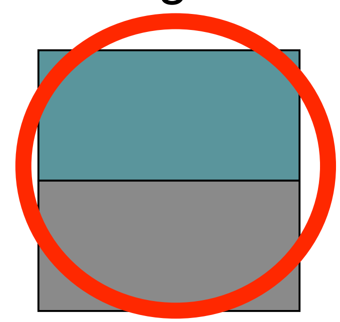
Sample Partitioning: POP

- 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.

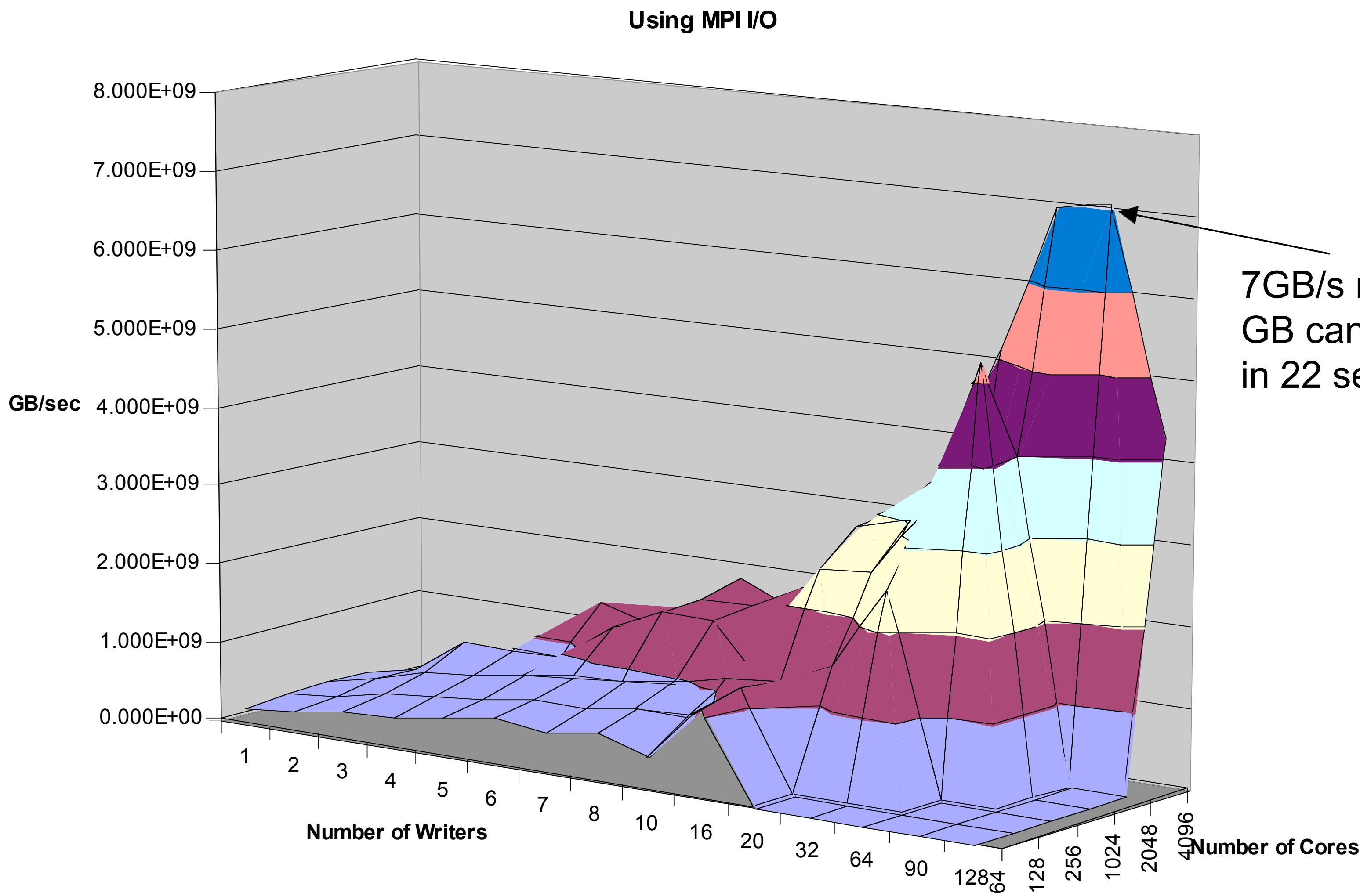
Sample Partitioning: POP

- Given a nearly square partitioning, the number of nodes simultaneously performing IO is approximately the square root of the total number of compute nodes.
  - 2500 compute nodes - 50 IO nodes
  - 10000 compute nodes - 100IO nodes
  - 25600 compute nodes - 160 IO nodes
- Many partitions allow a reasonable assignment of ionodes

For Example:

- An array of 8 byte reals (300, 400, 40) on each of 10000 nodes
  - 4.8 million elements on each node
  - 48 billion elements total
  - 384 gigabytes data
  - 50 - 100 seconds to read or write at 4 - 8 gbyte/sec
  - 100 IO nodes

A Subset of Writers Benchmark



7GB/s means 157 GB can be written in 22 sec

Benchmark 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
- Real Application Data in the works, Come to CUG

Subsetting Example 2

- Jan test on XT3 with 1.5.29 (non-dedicated test), 96 OSTs
- Custom code (used earlier for scaling plot)
  - 1 file per proc (stripe width 1); 8640 processes (cores)
  - Will have it use a subset of the procs as ionodes
    - Can aggregate data or serially send data to ionodes
- Test1: 5 MB writes/reads (smaller buffer)
  - With 8640 writers
    - Writes: 1.4 GB/s; Reads: 2.0 GB/s (max of 2 runs)
  - With 960 writers, aggregating data on writers
    - Writes: 10.1 GB/s; Reads: 10.3 GB/s (max of 2 runs)
- Test2: 10 MB writes/reads (same buffer)
  - With 8640 writers
    - Writes: 2.3 GB/s; Reads: 3.1 GB/s (max of 2 runs)
  - With 960 writers, aggregating data on writers
    - Writes: 7.2 GB/s; Reads: 9.0 GB/s (max of 2 runs)

Subsetting Example 3

- XT4, 1.5.31, 144 OSTs, pgi/6.2.5
- Only Test (so far): 8 MB writes/reads
  - With 9216 writers
    - Writes: 619 MB/s (not as good as XT3)
  - With 1024 writers, serially sending data to writers
    - Writes: 10.4 GB/s

Take Home Notes

- 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
    - It works and requires no code changes
      - Can buffer stdout too
    - Loaded by default, “man iobuf” for more information
      - Typically can improve upon default settings
- 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 IO nodes (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 (need to verify it works on XT)

Take Home Notes (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
  - 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

Other

- Be aware of distribution of files across OSTs
  - If you do one file per process, make sure this distribution is equal across OSTs
    - lustre gets the distribution even or very close,
    - But if run with 8 procs, and then 16, and then ….
      - Sometimes you will not get even distribution across the OSTs
    - Even if you “replace” each file during a checkpoint, they will end up in the same OST
  - For small scale (< #ofOSTS), if one OST is used twice, it flatlines your scaling
  - Sometimes easiest to remove all files and then recreate them

Current Research

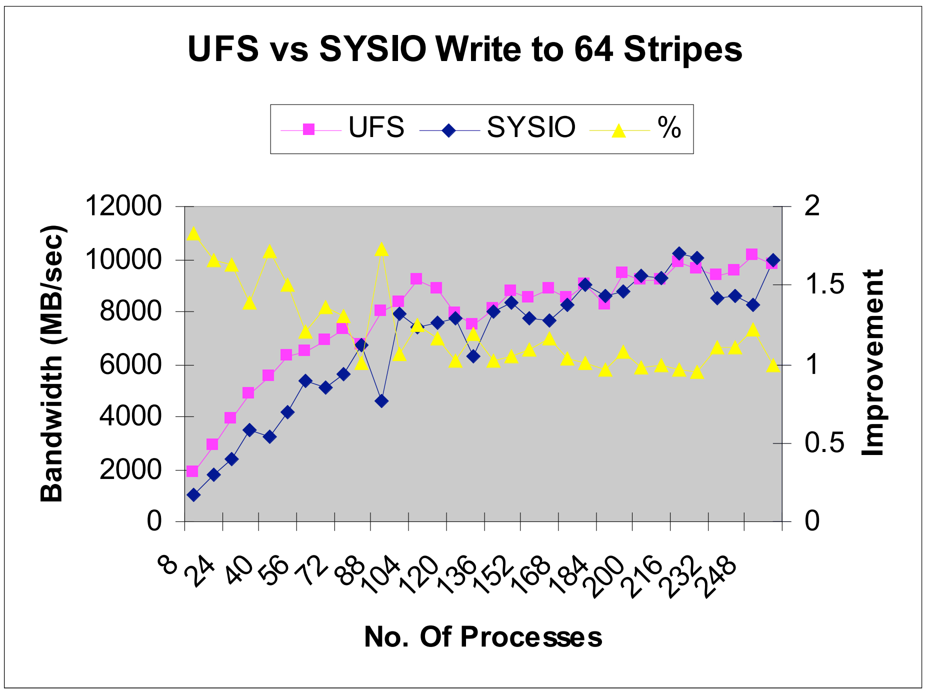
-

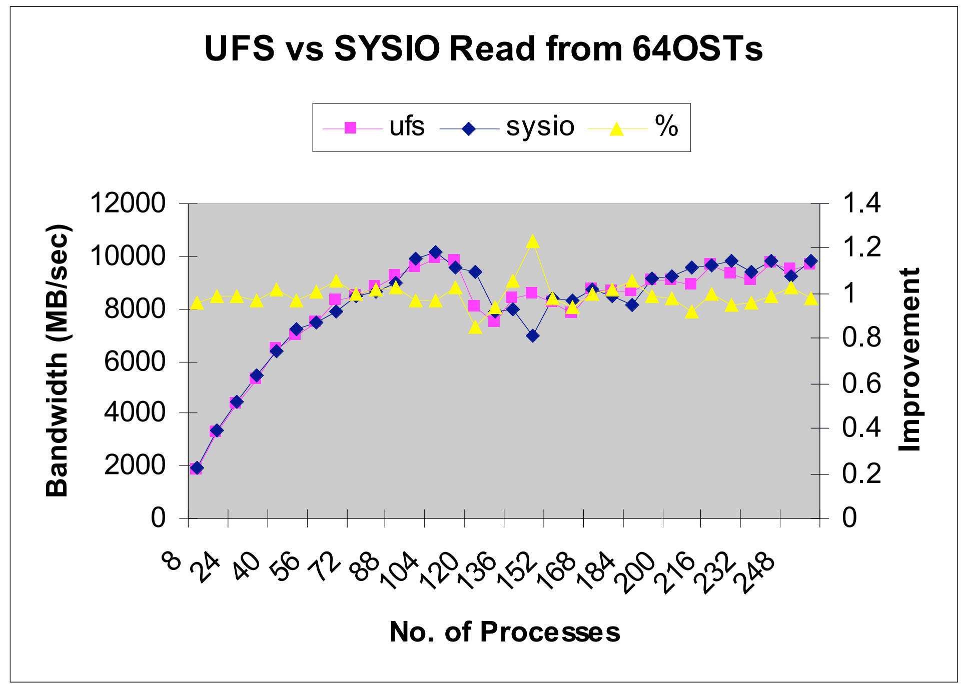
Parallel IO Instrumentation

- Default Parallel IO over XT3/4
  - ROMIO implementation over libsysio
  - Include Cray optimizations but proprietary code base
  - Hard for code dissection and performance analysis
- Creating a different parallel IO stack over XT3/4:
  - ROMIO over UFS
    - UFS-based ROMIO is applicable because Lustre is Posix compliant
  - Initial performance testing with IOR
  - Performance profiling with collective IO

ROMIO over UFS

- Performance with ROMIO over UFS
  - Write can be up 80% more efficient
  - Read is comparable, within 1%





-- Weikuan Yu at ORNL

Parallel IO Timing Profiling

- Why is collective IO slow?
  - Significant time spent in collective communication, and growing
  - What this tells us:
    - Communication is a scalability limiter inside collective IO
    - Do not forget hints to avoid collective communication if your output from large, contiguous, and non-overlapping regions

Timing Breakdown of Collective IO on XT

--Weikuan Yu at ORNL

Nprocs	Collective Comm	File IO	Comm/IO Ratio
IOR (millisec)			
16	**5210.87**	**22880.90**	**.23**
32	**13204.93**	**45290.90**	**.29**
64	**32034.95**	**89522.71**	**.36**
Flash IO Checkpoint file (millisec) – PNetCDF version			
16	**2872.84**	**2469.78**	**1.16**
32	**5696.86**	**4371.18**	**1.30**
64	**12019.0**	**8096.6**	**1.48**

What's Expected Soon?

- Upcoming Results
  - Attend CUG 2007 for Parallel IO stack efficiency over XT3/4
    - HDF5
    - Parallel NetCDF
    - MPI-IO
    - Fortran and Unix IO
  - With working examples on what tunables (hints) to use and how to use them over XT3/4 for these stacks.
- Upcoming optimizations
  - Exploit Lustre file joining, prototyped over Linux-based platforms
  - Explore overlapped communication and IO
  - Explore more scalable collective communication for IO

Documentation/help

- See Cray Docs at http://docs.cray.com
  - XT Programming Environment User's Guide
    - IOBUF and other buffering techniques
  - Lustre reference manual
  - Also see these man pages
    - Strided I/O functions: readx, writex, ireadx, iwritex
- See http://info.nccs.gov/resources/jaguar
  - **http://info.nccs.gov/resources/jaguar/iotips**
- **Much of this will be on the jaguar iotips page soon!**
- Contact your liaison or help@nccs.gov if you need help optimizing your IO

Acknowledgements

- XT architecture picture from “Cray and Lustre” talk by Carroll and Radovanovic at CUG06.
- Lustre architecture picture from “Lustre tutorial” given by R. Slick at CUG06.
- Lots of material taken from “Efficient I/O on the Cray XT” talk by J. Larkin at Cray Technical Workshop, Feb 07.
- The “Current Research” material provided by Weikuan Yu.