CRAY
THE SUPERCOMPUTER COMPANY

# Achieving I/O Performance

Kevin Roy

# Agenda

- Motivation
- I/O Infrastructure
  - Hardware
  - Software Layers
- I/O Strategies
  - Input
  - Output
- Achieving Performance

# Motivation

- Asking who is interested in I/O optimization people will fall into one (or more) camps:
  - It doesn't affect me – I compute for 12hrs on 8192 cores and to compute "42" thus IO is not important to me!
  - Disks are slow so there is nothing I can do about it so optimization is irrelevant
  - I do I/O but I have no idea how long it takes nor do I care.
  - I know I/O does not scale and I'm not here to fix it
  - I/O has never really been a problem until I got on this large Cray system
    - Oh, and I also upped my job size from 128p to 8192p....
  - I run for 12 hrs and it takes 20 minutes to create a checkpoint file and this seems insignificant.
  - If it is expensive I will do it less often.
  - My I/O works well – I dump my 2GB dataset in 2 minutes this is better than I see elsewhere.

# Answers

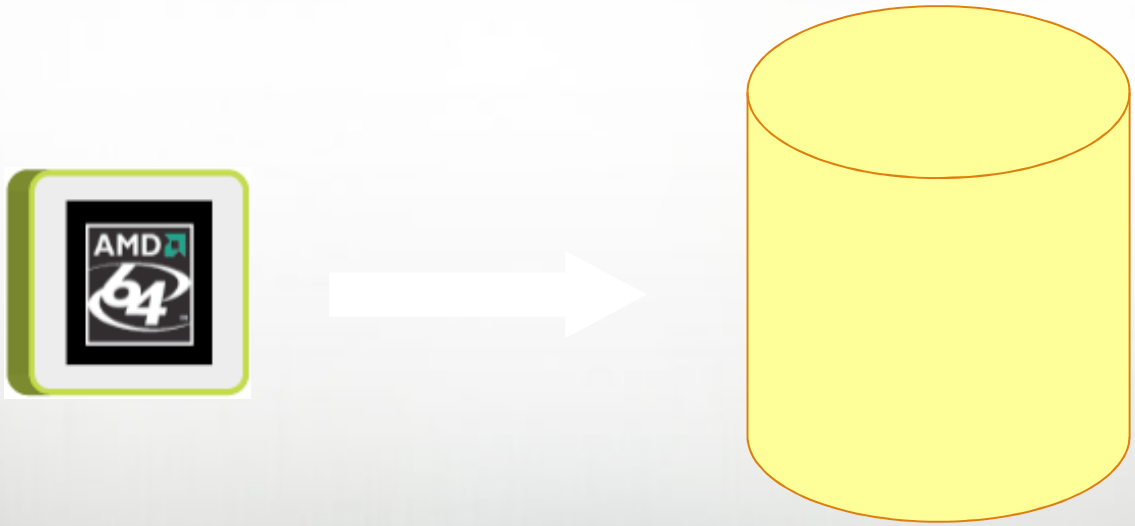
- Everyone should care
  - Either you affect everyone
  - Or others affect you.
- I/O is a shared resource unless the disk is a dedicated resource
  - On Cray XT4 no disk resource is dedicated – remember that /tmp is memory so not very big nor permanent.

# Targets

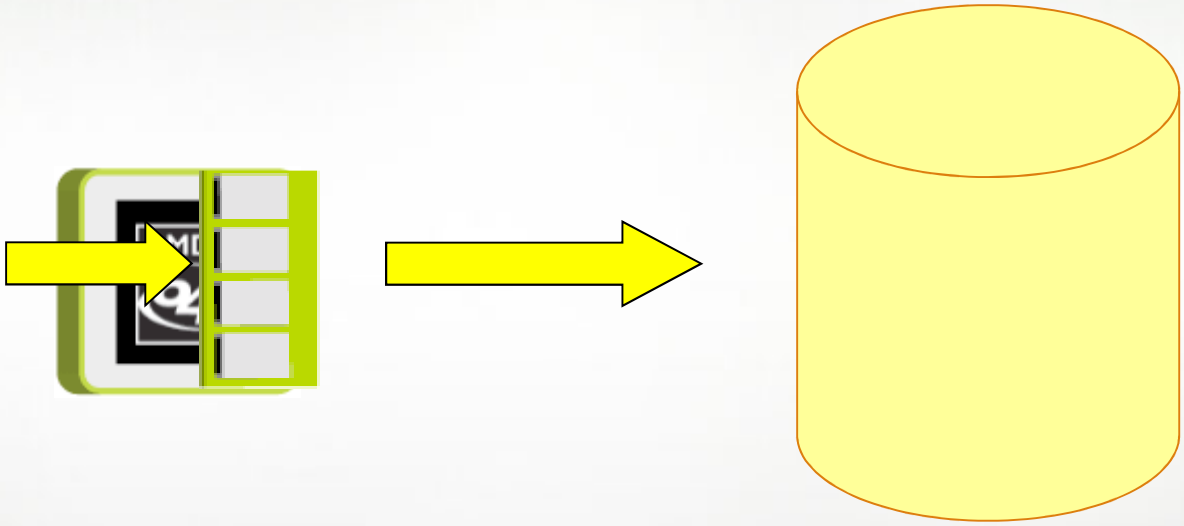
- I don't care about what order data reaches disk and how it is split. All that matters is performance
  - Good – measure in GB/s (maybe higher)
- Format and structure and portability matter but I've tried to make my code use large contiguous blocks
  - Measure I/O in 100's MB/s
- None of the above apply
  - 10's MB/s – maybe lower.
  - You should look at the I/O pattern in your code
- I have no control – I use an external library that I have no control over.
  - You always have control over how you use the library
  - Choose another library, an optimized version or a parallel version

# What is I/O

- Input is the need to load data into my program/data space
- Output is the need to move data out of my data space
  - This could either be from my program
  - Or to disk

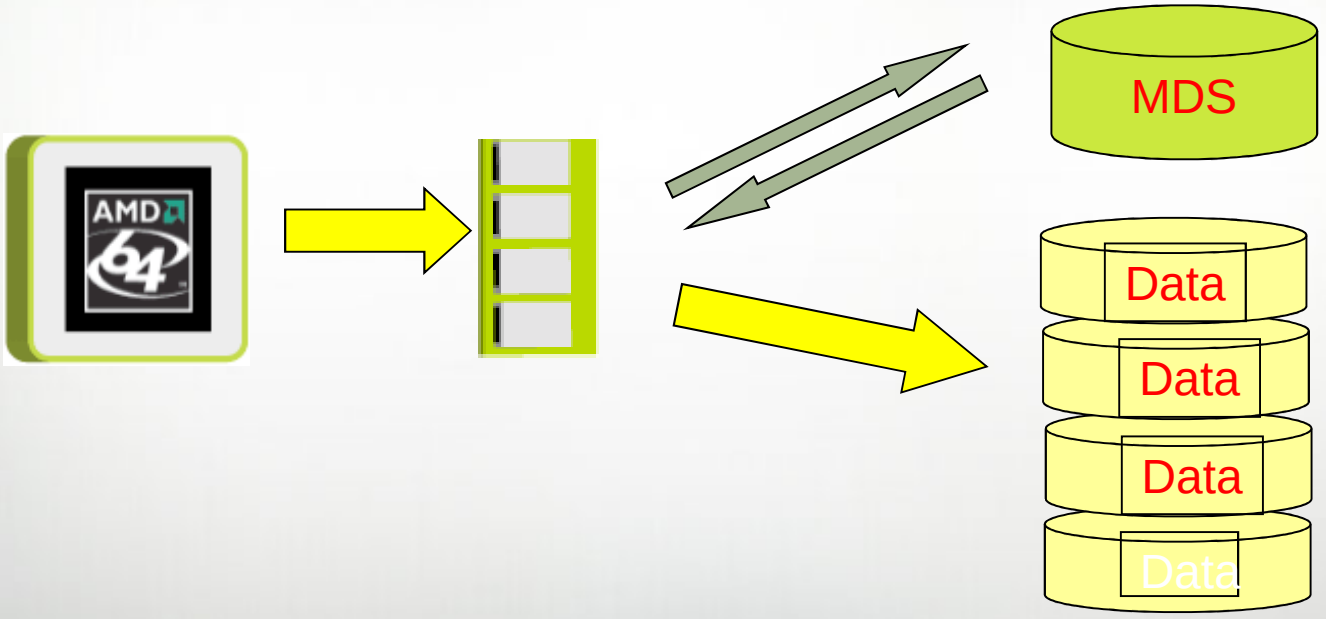


- That looks really simple but the real situation is:

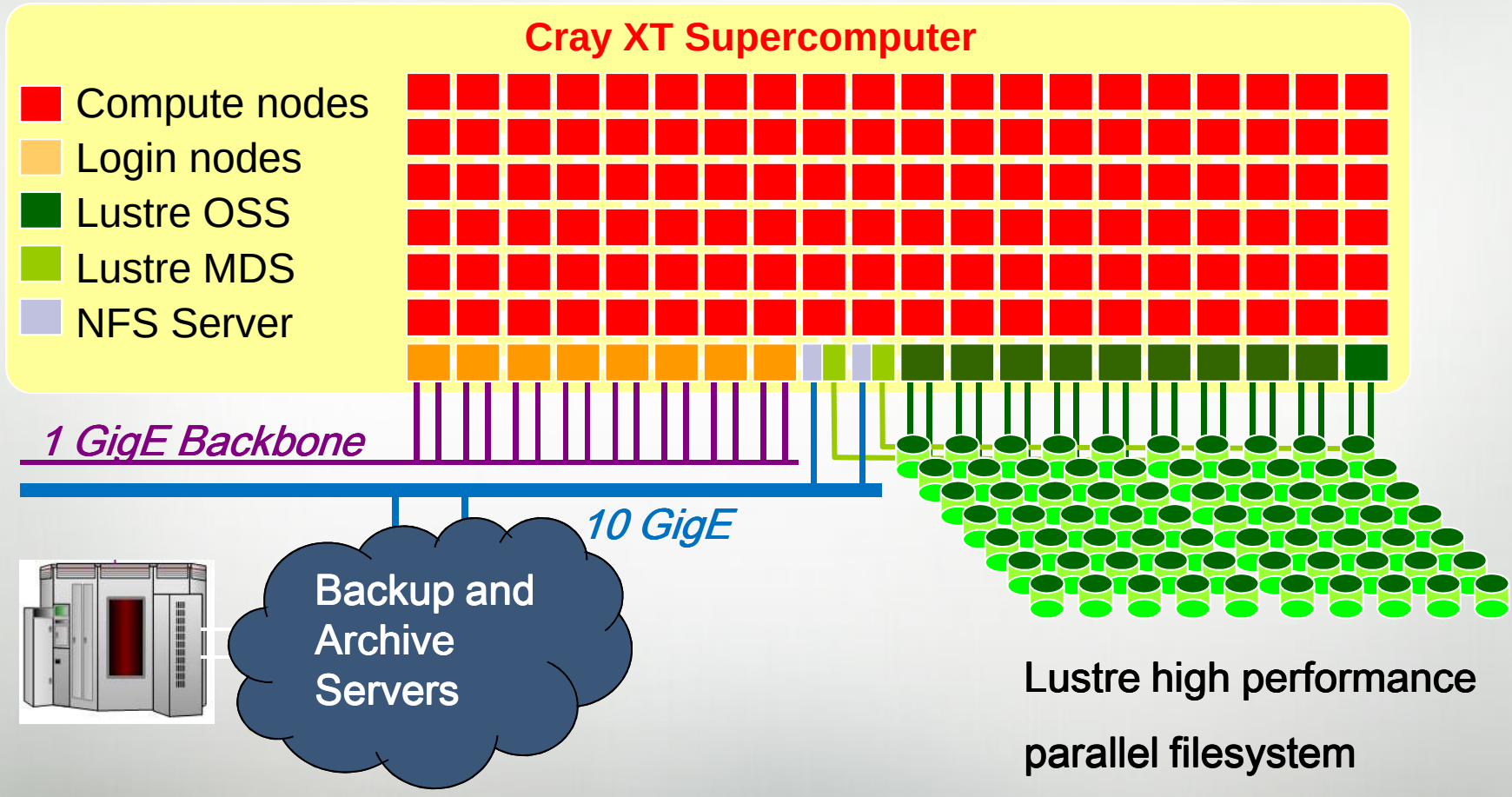


- Linux is really good at using buffer cache.
  - Much better than catamount

- Actually it is a little more complicated than that with Lustre…
  - The interaction with the disk consists of two phases



# Machine Layout



# Application I/O Strategies

1. Every rank outputs the data to a separate file
   a) GOOD – simple to program
   b) OK– restarting probably requires the same number of ranks
   c) GOOD – can be efficient at writing
   d) BAD – at reading
2. Data is collected to one rank and one file is stored
   a) GOOD – simple to program but it needs MPI to communicate the data
   b) BAD – Insufficient memory on one rank to cache the data in OS buffers so data has to go to disks
   c) BAD – All processors send messages to one rank and it has to send the data out. Bottleneck is the communications on one node
   d) VERY BAD – No parallelism, in fact due to overhead of communicating the data it is probably worse than serial

# Application I/O Strategies

3. Every rank does MPI-IO
   a) GOOD – portable
   b) OK – Can be more difficult to program than the above methods
   c) GOOD – someone else can optimize the MPI-IO library
   d) GOOD – configurable options
4. Using an I/O server approach
   a) GOOD – portable
   b) OK – needs some work to rewrite how the I/O is performed.
   c) GOOD – can take advantage of large numbers or cores available on nodes
   d) GOOD – asynchronous. 1TB checkpoint data set co[illegible] minutes or more to create.

# Achieving Performance

# I/O Parallelisation Opportunities

- Using Lustre presents many opportunities and facilities for parallelism
- It is important to understand them in order to take the best advantage

- There is parallelism
  - In data creation (this is done by compute nodes)
  - Parallel data paths out of the application
  - There are parallel paths into the I/O servers
  - The I/O servers are parallel using RAID file systems

# Buffering

- There is buffering at most levels
  - MPI has buffers
  - The compute node performing the I/O
  - Fortran runtime has buffers
  - The OSTs have buffers
- Some of which are configurable
- It is important to use the buffering effectively
- If you do large efficient I/O buffering adds extra layers which are not needed.
- Small requests should use a buffering layer to collect the small requests into larger requests.

# Increasing Lustre Performance

- This is covered in greater depth later but in order to get a flavour of Lustre performance …
- We apply attributes to files or directories
  - For directories the attributes apply to all files contained in it
- We can describe
  - A stripe size
  - A stripe count (how many lustre nodes to spread a file across)
- As a quick test:
  - we can create two directories
  - Apply “lfs setstripe 0 -1 16” to one of them
  - Create two identical files and put one in each directory
  - In each directory simply copy the file to a new file name and measure the performance

# Increasing Lustre Performance

- The previous example shows good speed up if the file is large
- For small files this will not be the case
- For many files this may not always be the case

- Stripe size
  - If we have a 4MB write statement written to a file with a 1MB stripe size with a count of 4+ stripes the I/O uses 4 stripes to achieve parallelism

# Parallel Lustre Stripes

# Measure Your Data

- Is your I/O strategy a parallel one?
- What is your limiting factor?
  - Bandwidth from a single node?
  - Granularity of write requests?
  - Time taken to perform the I/O?
- Use the tools

# I/O Measurement with CrayPAT

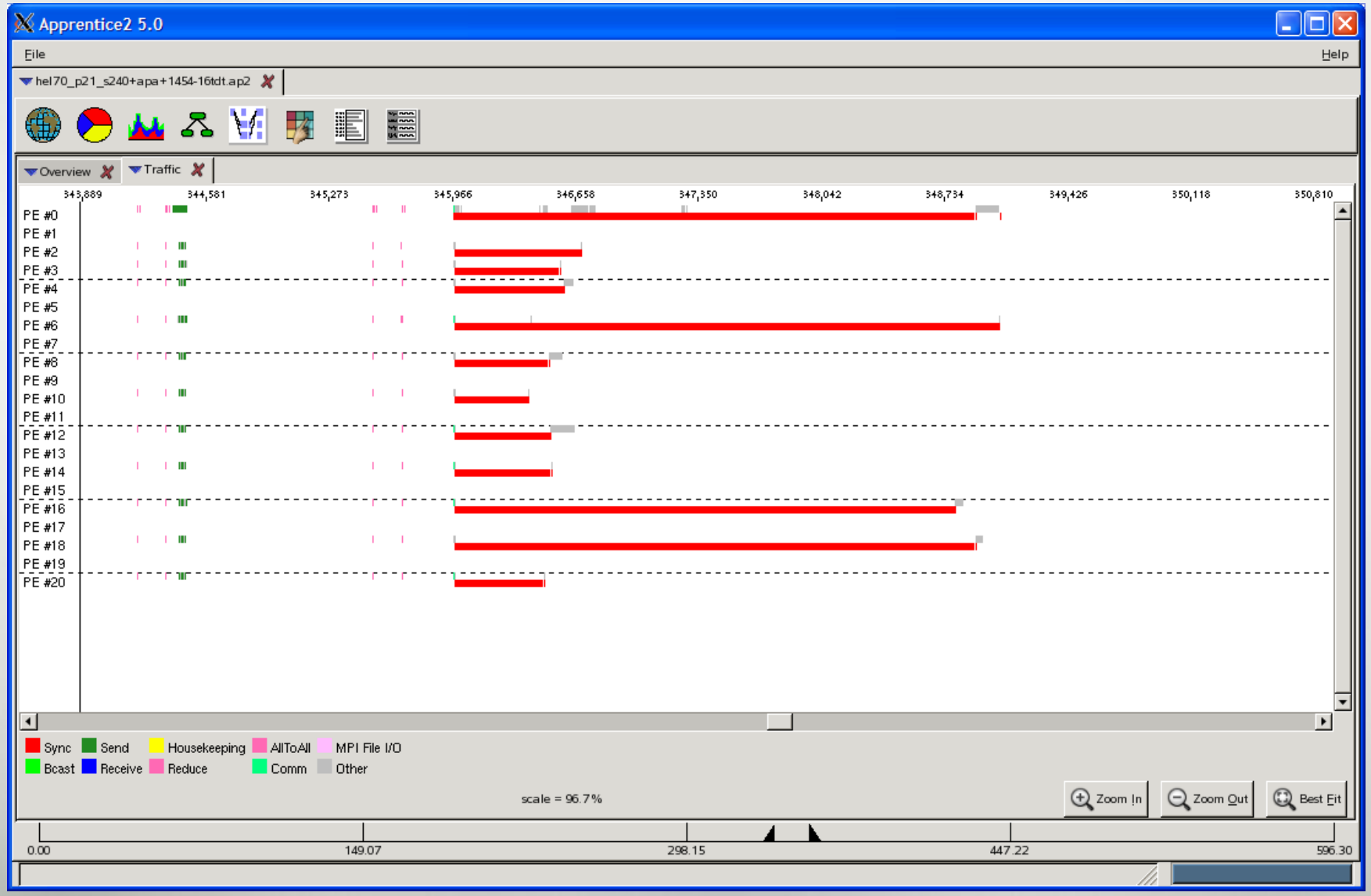
```
Table 7:  File Output Stats by Filename

Write Time |    Write MB | Write Rate |      Writes |Write B/Call |File Name
           |             |     MB/sec |             |             | PE[mmm]
           |             |            |             |             |  File Desc
 44.933754 | 2936.514680 |  65.352089 | 1847.000000 |  1667113.60 |Total
|-------------------------------------------------------------------------------
|  2.864199 |   93.251465 |  32.557611 |   24.000000 |  4074218.67 |./state/f000000
||------------------------------------------------------------------------------
||  2.864199 |   93.251465 |  32.557611 |   24.000000 |  4074218.67 |pe.0
3|          |             |            |             |             | fd.20
||  0.000000 |          -- |         -- |          -- |          -- |pe.3
||  0.000000 |          -- |         -- |          -- |          -- |pe.5
||==============================================================================
|  2.714276 |   93.251465 |  34.355926 |   24.000000 |  4074218.67 |./state/f000010
||------------------------------------------------------------------------------
||  2.714276 |   93.251465 |  34.355926 |   24.000000 |  4074218.67 |pe.10
3|          |             |            |             |             | fd.10
||  0.000000 |          -- |         -- |          -- |          -- |pe.6
||  0.000000 |          -- |         -- |          -- |          -- |pe.5
```

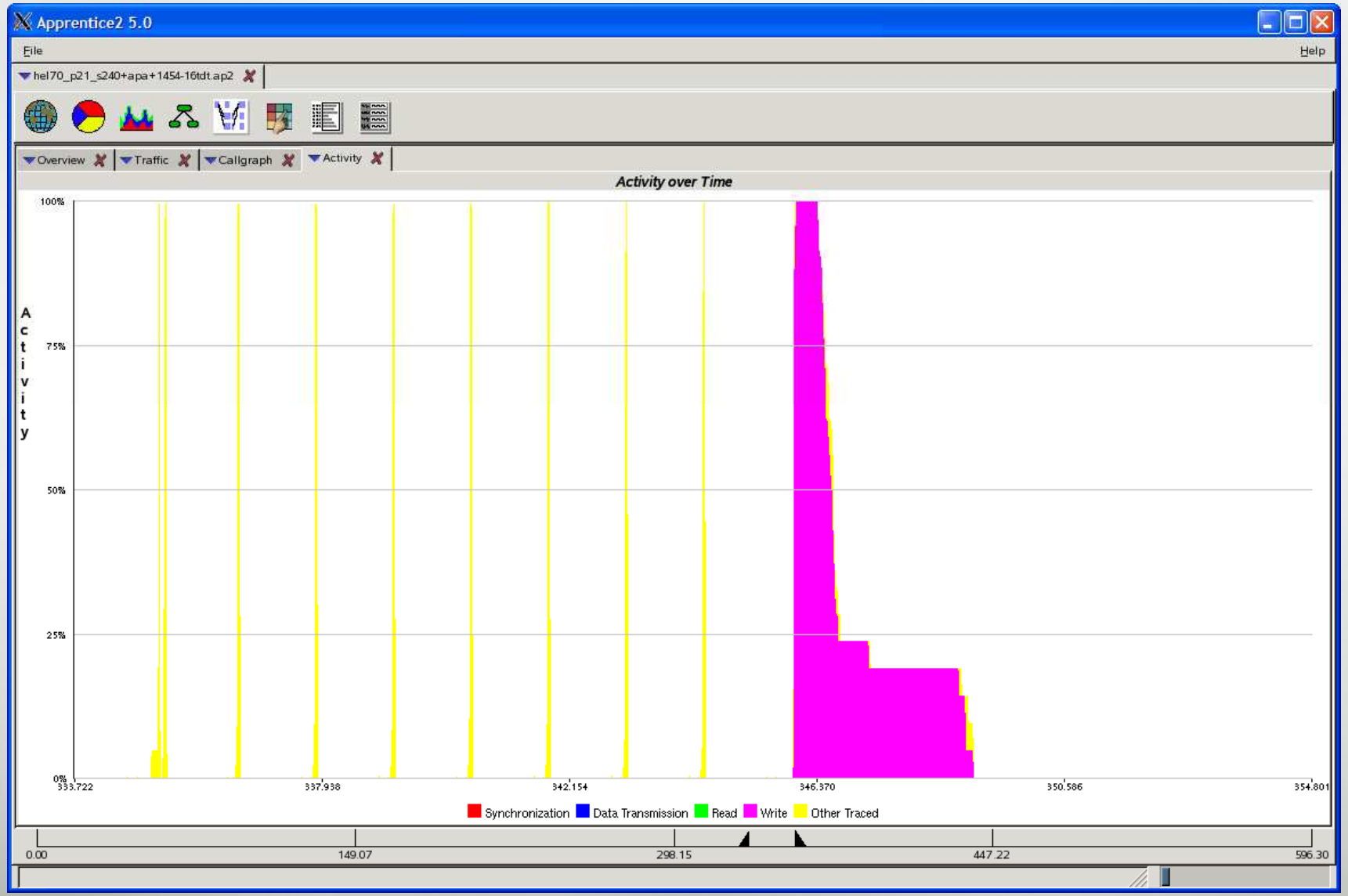
# I/O Measurement with CrayPAT

```
||=============================================================================
|  2.080276 |   93.251465 |  44.826483 |   24.000000 |  4074218.67 |./state/f000001
||-----------------------------------------------------------------------------
||  2.080276 |   93.251465 |  44.826483 |   24.000000 |  4074218.67 |pe.1
3|          |             |            |             |             | fd.13
||  0.000000 |          -- |         -- |          -- |          -- |pe.3
||  0.000000 |          -- |         -- |          -- |          -- |pe.5
||=============================================================================
|  1.844042 |   93.251465 |  50.569048 |   24.000000 |  4074218.67 |./state/f000020
||-----------------------------------------------------------------------------
||  1.844042 |   93.251465 |  50.569048 |   24.000000 |  4074218.67 |pe.20
3|          |             |            |             |             | fd.13
||  0.000000 |          -- |         -- |          -- |          -- |pe.6
||  0.000000 |          -- |         -- |          -- |          -- |pe.5
||=============================================================================
|  1.830046 |   93.251465 |  50.955807 |   24.000000 |  4074218.67 |./state/f000009
||-----------------------------------------------------------------------------
||  1.830046 |   93.251465 |  50.955807 |   24.000000 |  4074218.67 |pe.9
3|          |             |            |             |             | fd.12
||  0.000000 |          -- |         -- |          -- |          -- |pe.6
||  0.000000 |          -- |         -- |          -- |          -- |pe.5
```

# Apprentice 2 - Timeline



# Apprentice 2 – Activity Chart



# First Steps to Improving I/O Performance

- Understand your what your application needs and what your system can provide
  - How much data I am writing and how often?
  - What's the peak bandwidth of the system?
  - Ok, what's the real bandwidth?
- Determine where you have a problem
  - Am I performing a lot of small writes that could be combined?
  - Am I overwhelming the FS with too much at once?
  - Do I really need to save all of this data every single timestep?
- Try different Lustre parameters
  - Could more or fewer OSTs help?
  - Can I improve performance with larger stripes?
- If possible, make a small test program out of the I/O portion of your code
  - Sometimes it's easier to test parameters with a smaller kernel than a full application
- Seek help

# Using MPI I/O Hints

- The MPI specification provides a way to give “hints” to the MPI-I/O layer for better performance.
- Hints to try
  - cb_nodes -> Built-in subsetting
  - cb_read/write -> Enable/Disable “collective buffering”
  - cb_buffer_size -> Controls the size of the intermediate buffer used in collective buffering
  - ds_read/write -> Enable/Disable “Data sieving”, used with non-contiguous I/O requests
    - Generally not recommended
  - direct_io -> Enables direct I/O, bypassing kernel buffers
    - Requires xt-mpt/3.0.0.8 (pre-release as of April 3, 2008)
    - Requires data buffer to be aligned to page boundary
    - Can help with very large I/O requests
- Can be set via API or via MPICH_MPIIO_HINTS environment variable
  - Example: export MPICH_MPIIO_HINTS="${FILE}:direct_io=true:romio_cb_read=disable:romio_cb_write=disable:romio_ds_read=disable:romio_ds_write=disable"
- Setting MPICH_MPIIO_HINTS_DISPLAY=1 will print your MPI-IO hints when a program is run.

# Best Practices

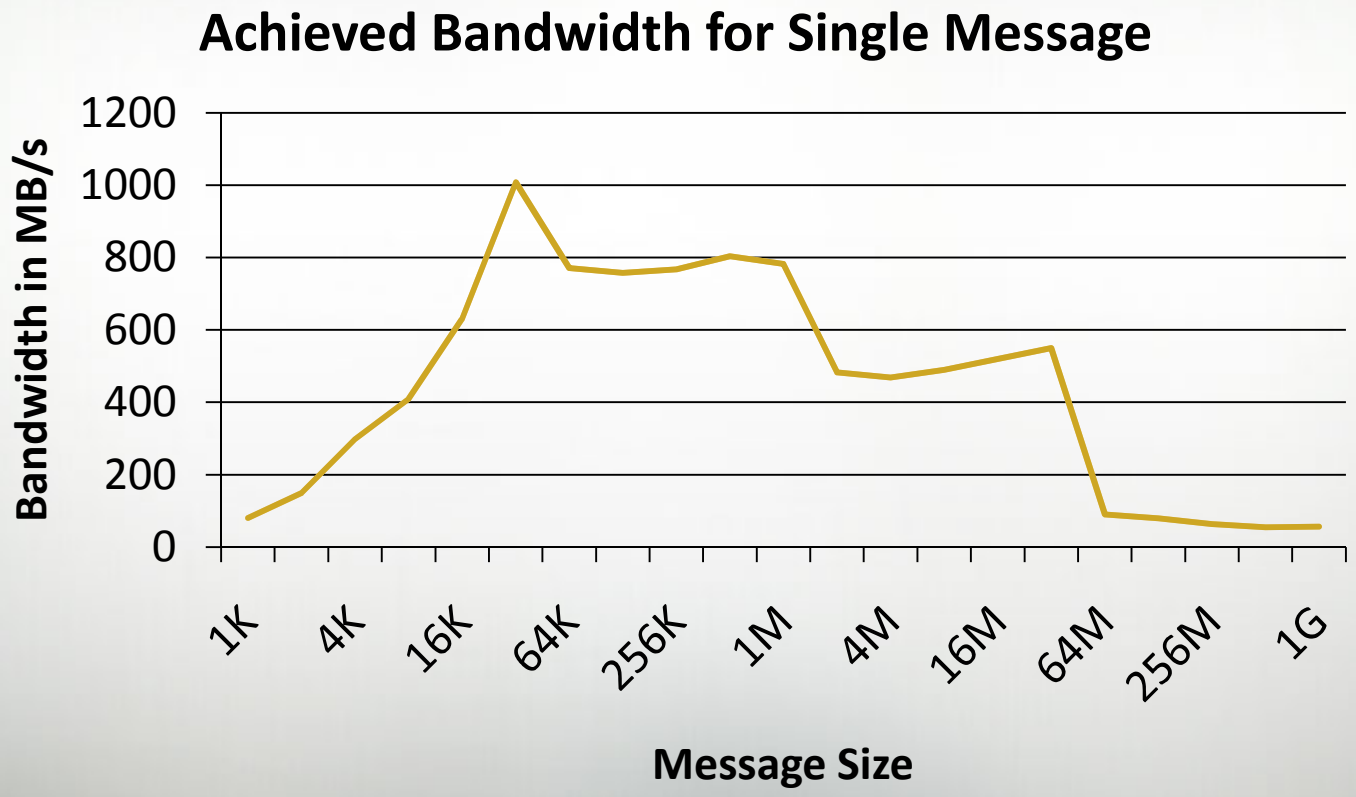
- Remember, this is not your laptop, I/O for HPC has many challenges
  - Unfortunately, I/O rarely scales at the same rate as FLOPS
- Do not open a file from hundreds/thousands of nodes at the same time
  - Metadata operations are slow, do them infrequently
  - Too many will overwhelms the MDS at very large scales
- Do not try to do all of your I/O through 1 node, unless you have a little data or a lot time.
  - Unable to saturate bandwidth
- Do not do I/O from every node for nodes over ~1K processes
  - Performance degradation
  - Where this degradation occurs varies by system

# Best Practices (cont.)

- Buffer so that you can do large I/O operations
  - Bigger writes/reads perform better
  - Subsetting can help improve buffering
- Many files can perform better than 1, but can be less convenient
  - Try doing operations on many thousands of files, non-trivial
  - As discussed, too many at once files can lead to MDS overload
    - Subsetting can help with this too!
- Stripe Appropriately to Your I/O
  - Many nodes to individual files: Stripe to 1 OST
  - Many nodes to 1 file: Stripe to many OSTs
  - 1 node to 1 file: Stripe to few OSTs
  - Set your stripe size to larger than 1MB, 4MB suggested

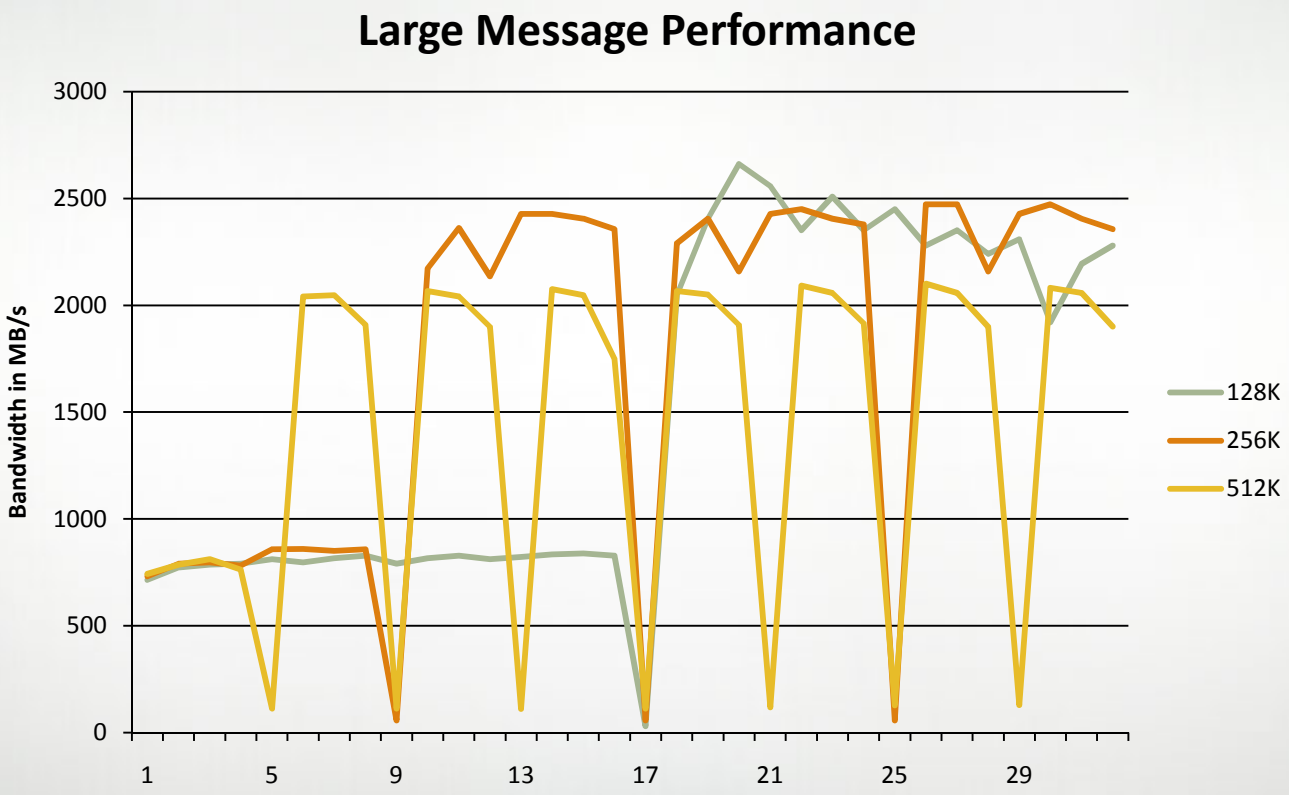
# Serial Performance

- Basic IO measurement. This is symptomatic of the spokesperson method, where data is accumulated to one rank and sent to disk as one big message.



# Experiment III – Per message bandwidth

- But what about larger messages




- Looks like a buffer issue when it hits 2MB

CRAY
THE SUPERCOMPUTER COMPANY

# Running on XT Compute Nodes

# Questions / Comments
# Thank You!