Lecture 18
I/O Performance and Checkpoints

EN 600.320/420/620

Instructor: Randal Burns

4 November 2020

Department of Computer Science, *Johns Hopkins University*

The I/O Crisis in HPC

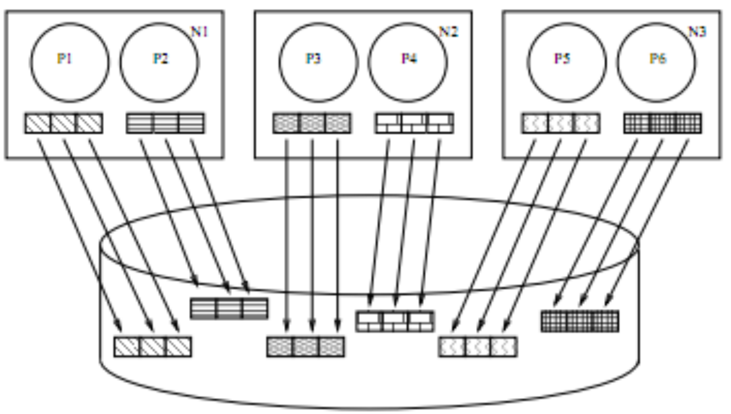
In a world where FLOPS is the commodity…….

…..Disk I/O often limits performance

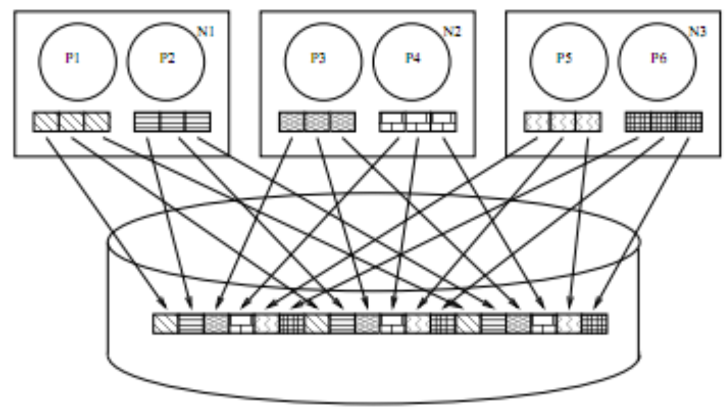
- Any persistent data must make it off the supercomputer
  - To magnetic disk or solid state storage
- Storage is not as connected to the high-speed network as compute
  - Because it needs to be shared with other computers
  - Because it doesn't add to TOP500 benchmarks

Where does the I/O Come From?

- Checkpointing!
  - And, writing output from simulation (which is checkpointing)
- Checkpoint workload
  - Every node node writes local state to a shared file system
  - Using POSIX calls (FS parallelized) or MPI I/O



(a) N-N



(c) N-1 strided

J. Bent et al. PLFS: A Checkpoint File Systems for Parallel Applications. SC, 2009.

Why Checkpointing

- At scale failures occur inevitably
  - MPI synchronous model means that a failure breaks the code
  - Lose all work since start (or restart)
- Each checkpoint provides a restart point
  - Limits exposure, loss of work to last checkpoint
- By policy, all codes that run at scale on supercomputers MUST checkpoint!
  - HPC centers want codes to do useful work

Checkpoint Approaches

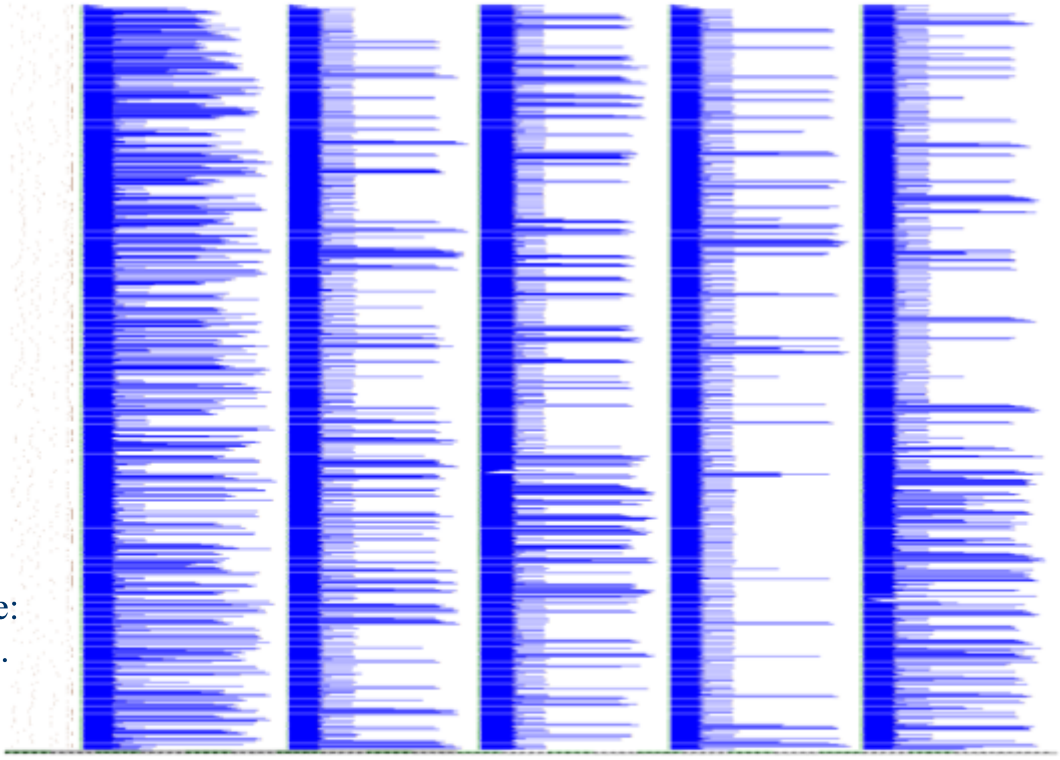
- Automatic: store contents of memory and program counters
  - Brute force, large data, inefficient
  - But easy, no development effort
  - New interest in this approach with the emergence of VMs and containers in HPC.
- Application specific: keep data structures and metadata representing current progress. Hand coded by developer.
  - Smaller, faster, preferred, but tedious.
  - Almost all "good" codes have application specific checkpoints

A Checkpoint Workload

- IOR benchmark
  - Each node transfers 512 MB

- *How much parallelism?*
- *What effects?*

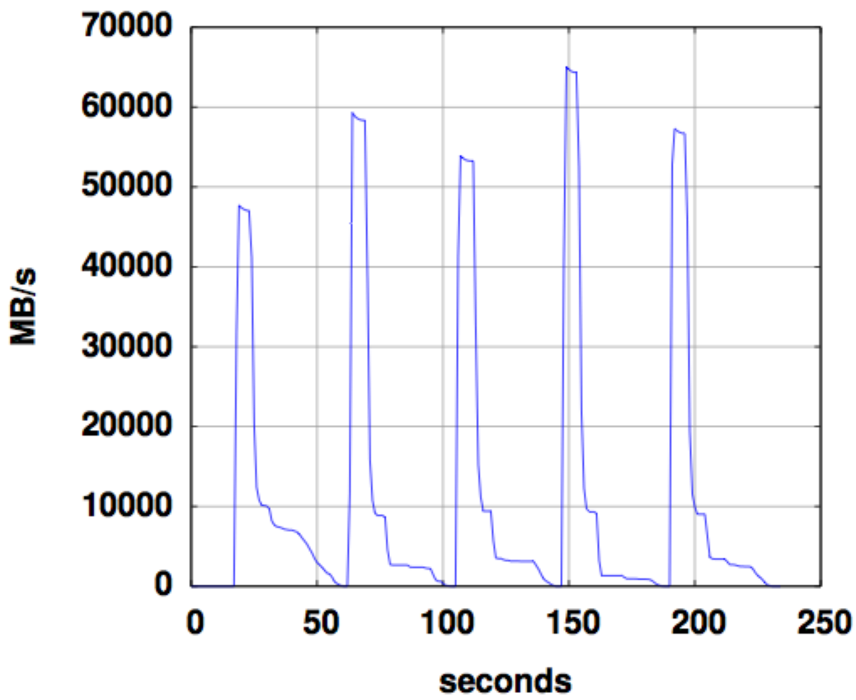
M. Uselton et al. Parallel I/O Performance: From Events to Ensembles. IPDPS, 2010.



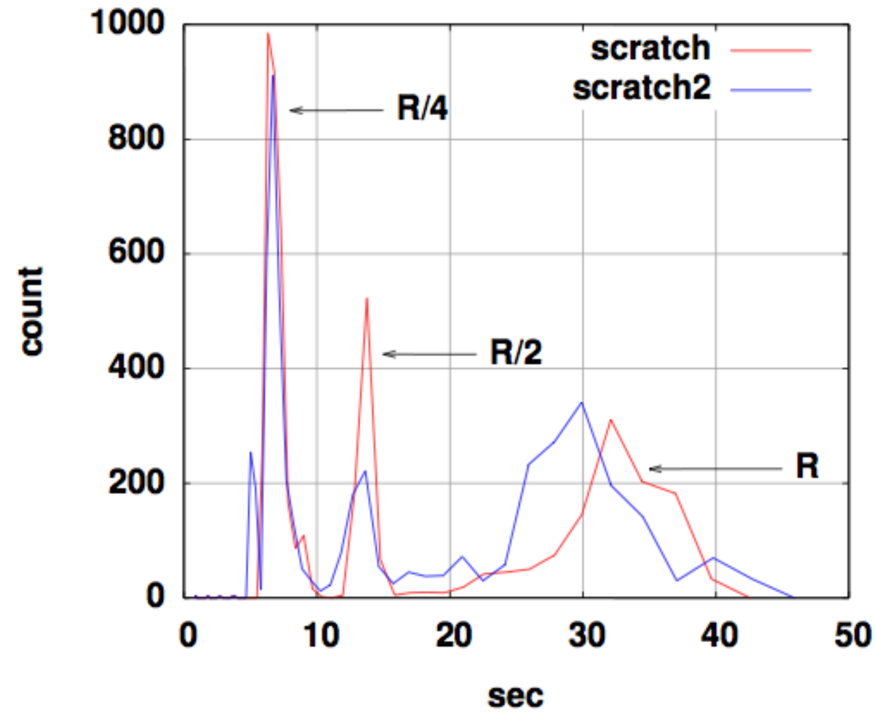
(a) I/O trace diagram

I/O Rates and PDF

- *What features do you observe?*



(b) Aggregate I/O rate

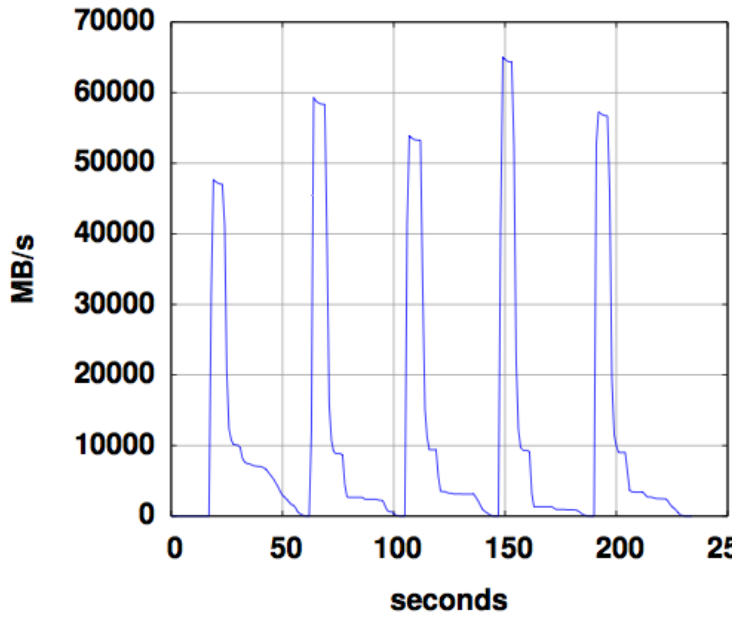


(c) I/O time distribution

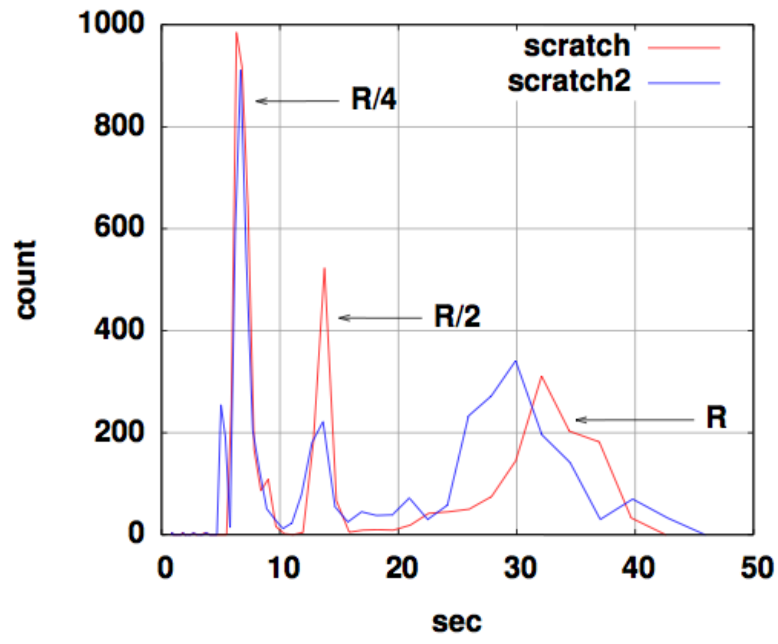
M. Uselton et al. Parallel I/O Performance: From Events to Ensembles. IPDPS, 2010.

I/O Rates and PDF

- *What features do you observe?*
  - Lagging processes = not realizing peak I/O performance
  - Harmonics in I/O distribution = unfair resource sharing



(b) Aggregate I/O rate



(c) I/O time distribution

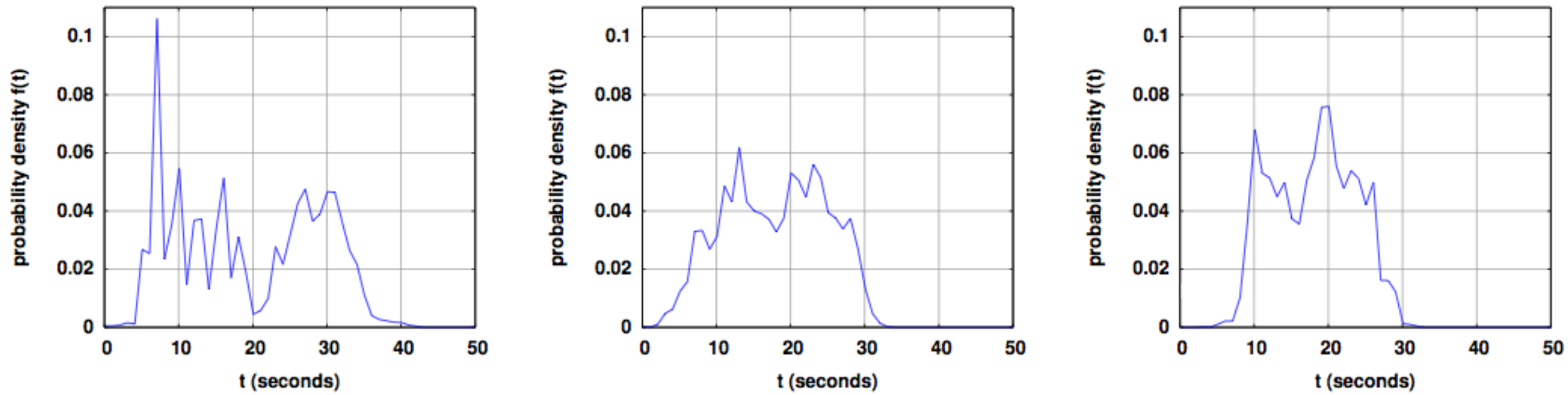
M. Uselton et al. Parallel I/O Performance: From Events to Ensembles. IPDPS, 2010.

Statistical Observations

- Order statistics
  - Fancy way of saying, the longest operation dominates overall performance
- Law of large numbers
  - I don't think that they make this analysis cogent
  - It's right, but Gaussian distribution is not what matters
  - A better, intuitive conclusion is
- (RB interprets) smaller files are better
  - The worst case slow down on a smaller transfer takes less absolute time than on a large transfer
  - As long as transfers are "big enough" to amortize startups costs

Smaller Files Improve Performance

- Non-intuitive
  - Smaller operations seems like more overhead
  - But, a property of statistical analysis
- Smaller better as long as fixed costs are amortized
  - Obviously, 1 byte is too small



(a) two 256 MB transfers (b) four 128 MB buffers (c) eight 64 MB transfers

Figure 2: IOR 512 MB transfer using 1024 processors where: a) 512 MB written via two 256MB `write()` calls. b) Four calls (128MB). c) Eight calls (64MB). Note that the distributions become progressively narrower and more Gaussian.

The Checkpoint Crisis

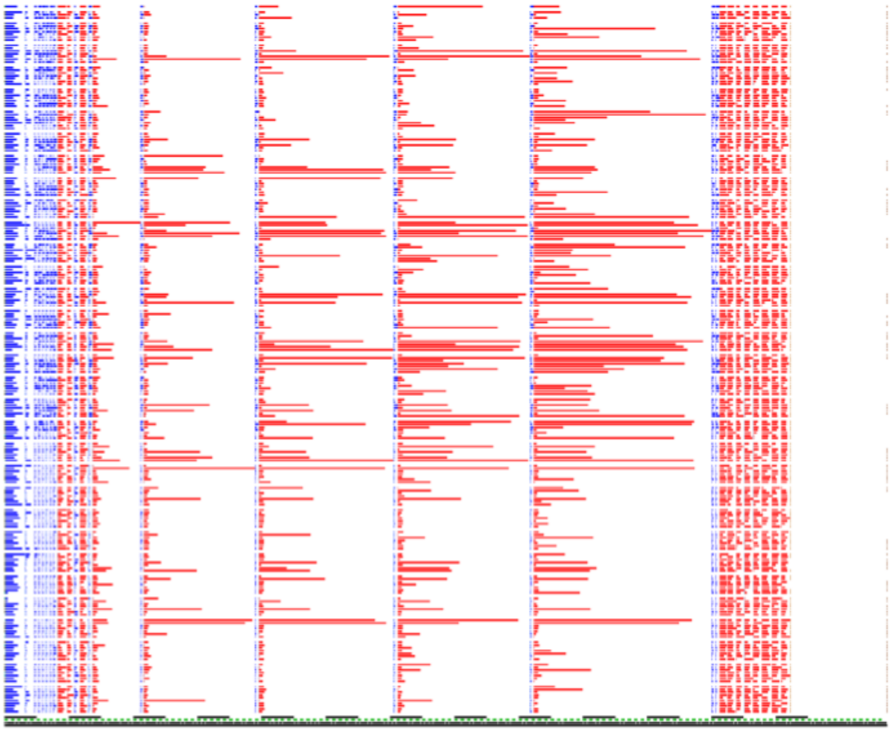
As HPC codes get larger, I/O becomes more critical

- Some observations
  - Checkpoint to protect against failure
  - More components increase failure probability
  - FLOPs grows faster than bandwidth
- Conclusion
  - Must take slower checkpoints more often
  - Eventually you will get no constructive work done between checkpoints
- Mitigation (just delaying the problem)
  - Burst buffers: fast (SSD) storage in high-speed network
  - Observe the checkpoint persistence is shorter than needed for output/analysis data

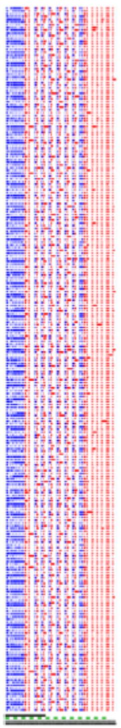
Extra Slides

Fixing I/O Performance

- Compare same I/O benchmark on two platforms
  - 256 nodes of Franklin and Jaguar

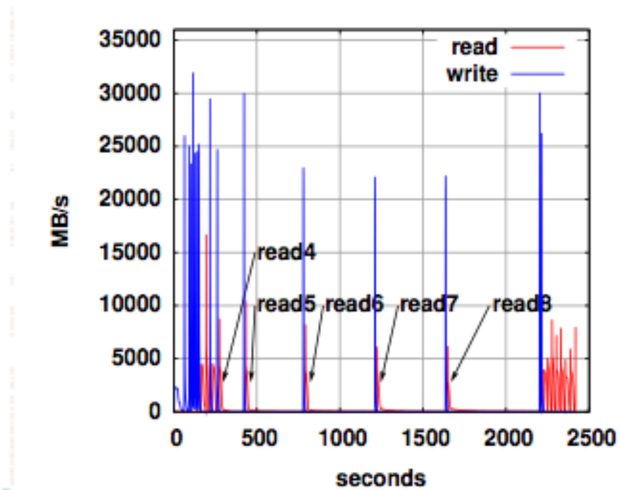


(a) Franklin trace

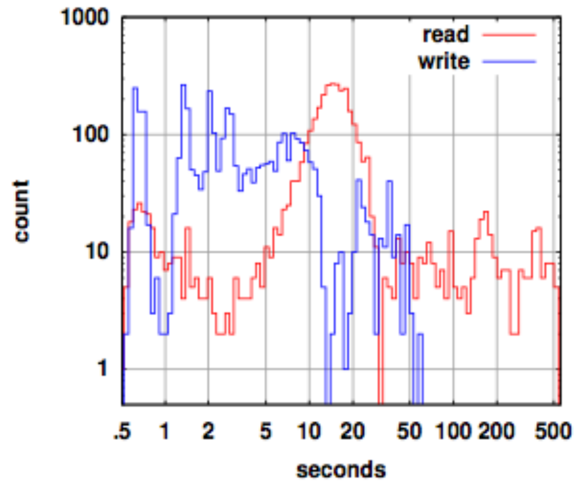


(d) Jaguar trace

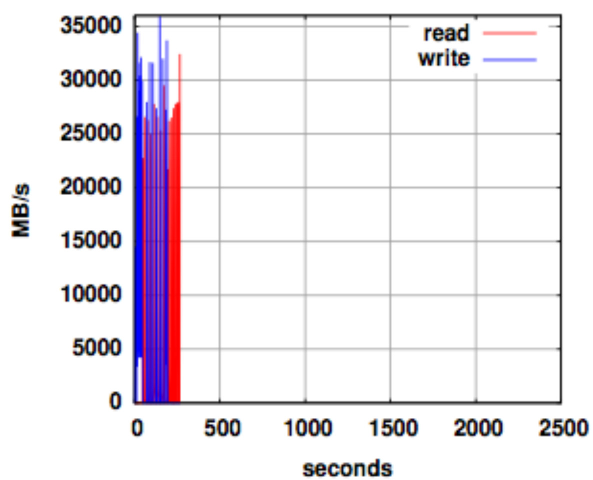
Problem = Long Read Delays



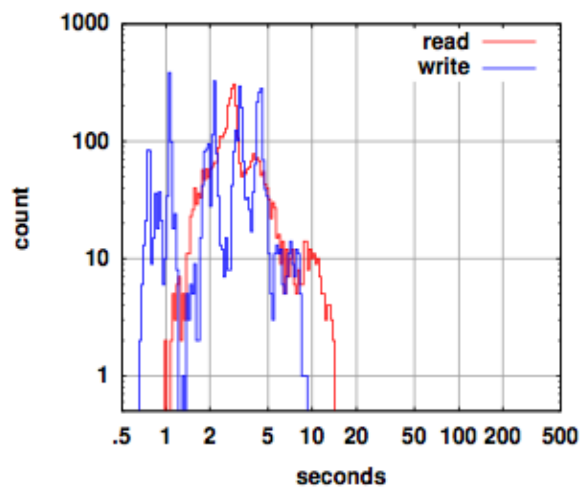
(b) Franklin aggregate I/O rate



(c) Franklin histogram



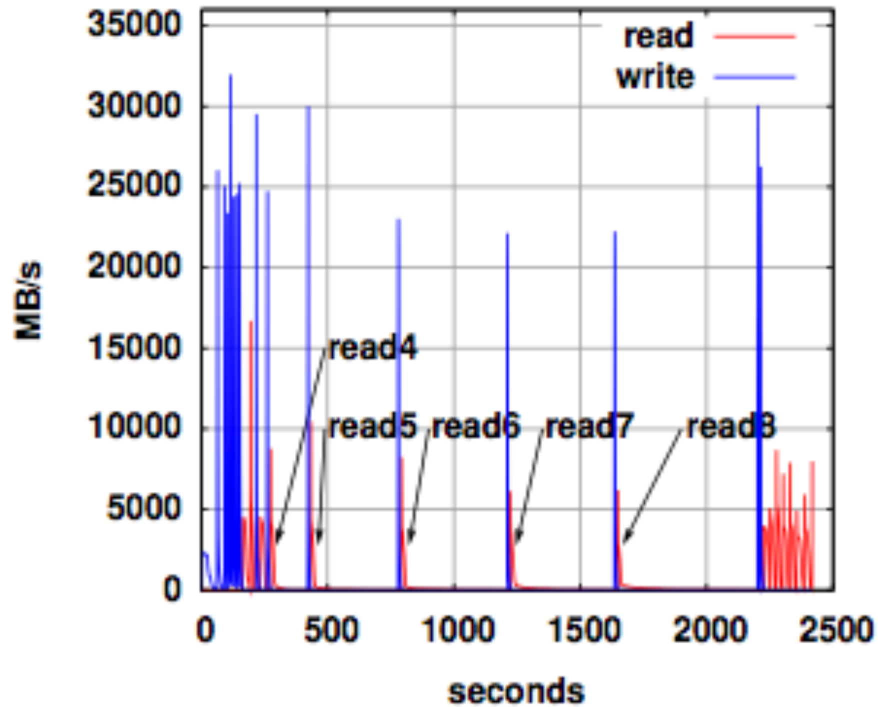
(e) Jaguar aggregate I/O rate



(f) Jaguar histogram

Problem Analysis

- Not all reads are slow
  - Just 4-8
- What special property do they have?
  - None: the reads are the same as earlier and later reads
- So, maybe something about ordering



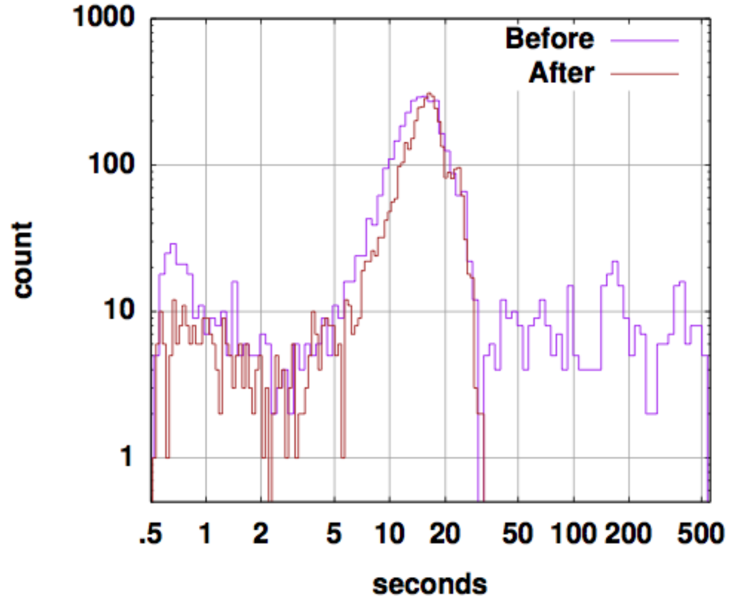
(b) Franklin aggregate I/O rate

Problem Analysis

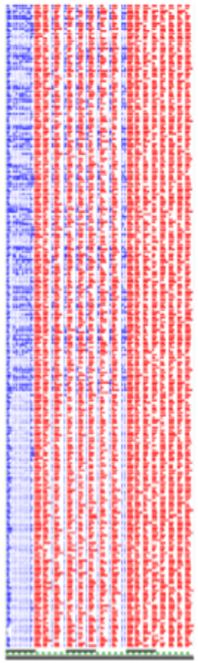
- After third read, system detects strided read pattern and performs read-ahead
  - Requires client side buffering of data
- Other uses of memory (client writes) consumed buffer space, preventing the read-ahead from working
- Lustre file system executed a fall-back code path
  - Perform small reads when no buffer space is available
  - Small reads are very inefficient

Problem Resolution

- Patch the file system
  - Turn off read-ahead in this case
- Problem solved (4x improvement)



(b) Read performance before and after middleware update



(c) Franklin trace after update

Another Code (Resolution Process)

- Reduce the number of tasks (10K -> 80) and have each task do many small I/Os
  - Variability reduction from more small I/Os
  - Reduce resource use and contention (fewer actors)
- Align the request size to file system parameters
  - Increase transfer rate
- Defer and aggregate metadata writes
  - Avoid lots of small updates

Thought on MPI Performance

- Visualization tools work and matter
  - Examples of 5x to 10x differences
- I/O is a huge component of performance
  - This is only trending up
  - Memory capacity and processor speed makes more data
  - Scale requires more frequent checkpoints
- HPC is a complex and fragile ecosystem
  - Many parameters and implementation subtleties
- Order statistics rule
  - Only as fast as the slowest member
  - This gets more problematic as we use more nodes
  - HW errors and SW misconfiguations on one node can ruin a cluster. Must diagnose!