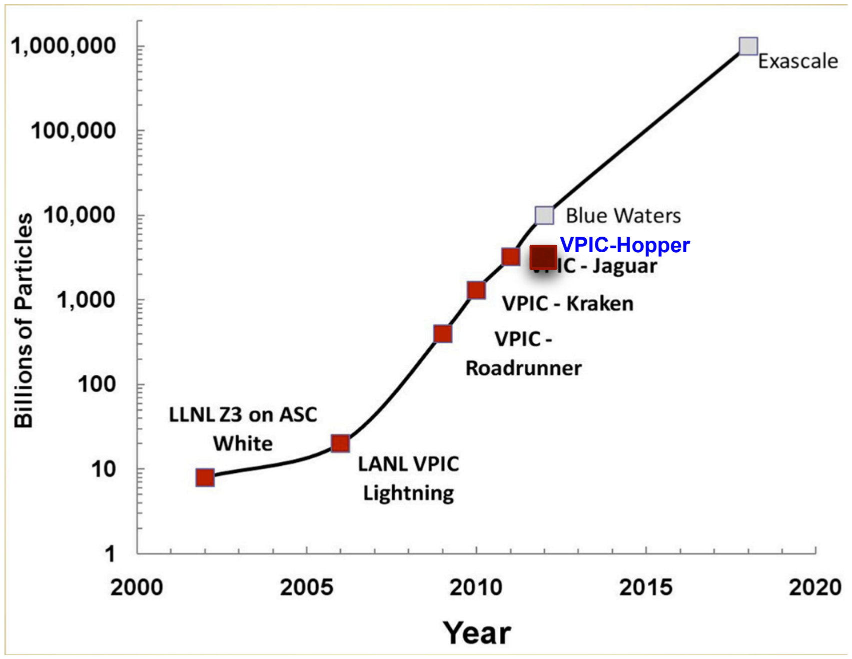
Trillion Particles, 120,000 cores, and 350 TBs: Lessons Learned from a Hero I/O Run on Hopper

Suren Byna (LBNL – Scientific Data Management group)

Andrew Uselton (NERSC), Prabhat (LBNL), David Knaak (Cray Inc.), Helen He (NERSC)

The application: Vector Particle-in-Cell (VPIC) Simulation

- ✧ A state-of-the-art 3D electromagnetic relativistic PIC plasma physics simulation
- ✧ It is an exascale problem and scales well on large systems
- ✧ An open boundary VPIC simulation of magnetic reconnection (Space weather)

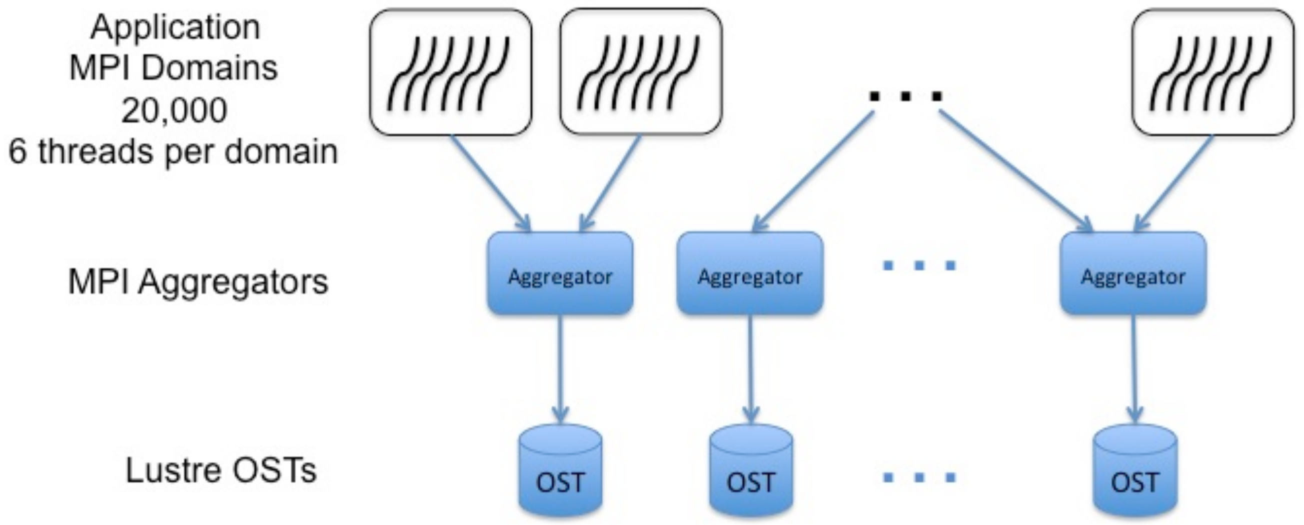


- ✧ NERSC Hopper Supercomputer
  - ○ 6,384 compute nodes; 2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node; 32 GB DDR3 1333-MHz memory per node; Interconnect with a 3D torus topology
  - ○ Lustre parallel file system with 156 OSTs at a peak BW of 35 GB/s

The application: Vector Particle-in-Cell (VPIC) Simulation

- March 1989: A power blackout in Canada affected 6 million people.
- October-November 2003: Solar panels fail on the $450 million Midori 2 research satellite, and astronauts take cover in the International Space Station.
- June 2011: Airlines report disruption of high-frequency radio communications near the Arctic.

VPIC Trillion Particle Simulation setup



- 20,000 MPI processes (MPI domains) using 120,000 cores
- Each MPI process writes ~51 Million (±15%) particles
- Each particle has 8 variables
- Lustre-aware MPI-IO implementation
  - ✓ MPI collective buffer size is equal to the stripe size
  - ✓ Number of MPI aggregators is equal to the stripe count
- **Particle dataset size varies (30TB to 43TB) per time step – A total of 350 TB data + 150 TB checkpoint data**

Data Challenges

- ✧ What is a scalable I/O strategy for storing massive particle data output?
  - o In situ analysis works well when analysis tasks are known *a priori*
  - o Many scientific applications require to store data for exploratory analysis
- ✧ What is a scalable strategy for conducting analysis on these datasets?
  - o Sift through large amounts of data looking for useful information
- ✧ What is the visualization strategy for examining the datasets?
  - o Display information that makes sense

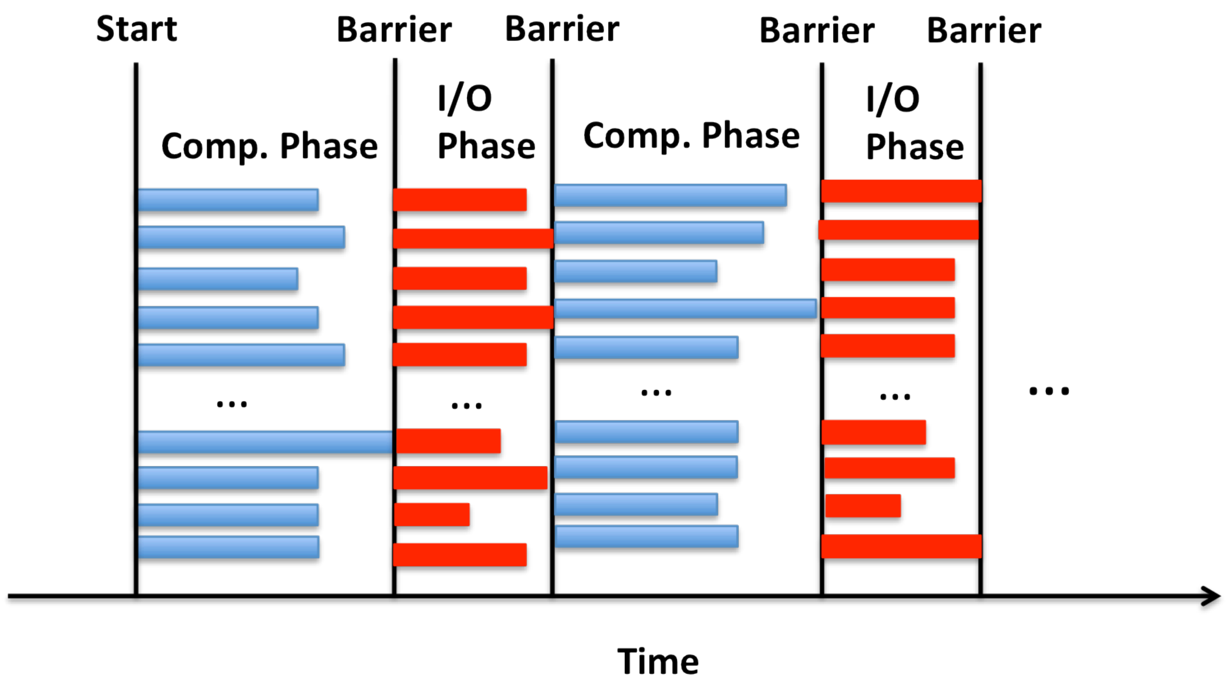
Expected challenges in running the application

- Scheduling a large job that would take ~80% of compute nodes
  - Queue time may be longer
  - Thanks to reg_xbig queue, which is turned on at 9PM on Fridays
- Lustre file system may be stressed due to large volume of data produced
- But…
  - There were a few more lessons we learned

Lessons Learned

1. Collective writes to a single shared HDF5 file can work as well as file-per-process writes
2. Tuning multiple layers of parallel I/O subsystem is a challenging task
3. Advance verification of file system hardware is important for obtaining peak performance
4. Advance verification of available resources for memory-intensive applications is important
5. Scalable tools are required for diagnosing software and hardware problems before running applications using 100k cores

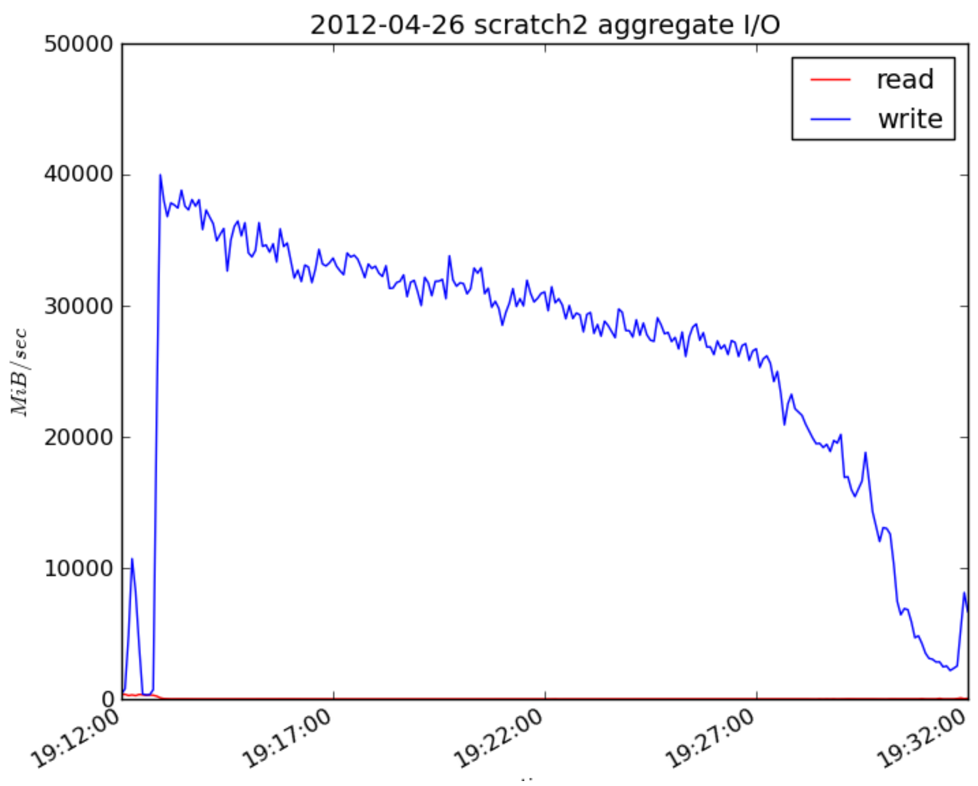
Lesson 1: Parallel HDF5 works – I/O pattern



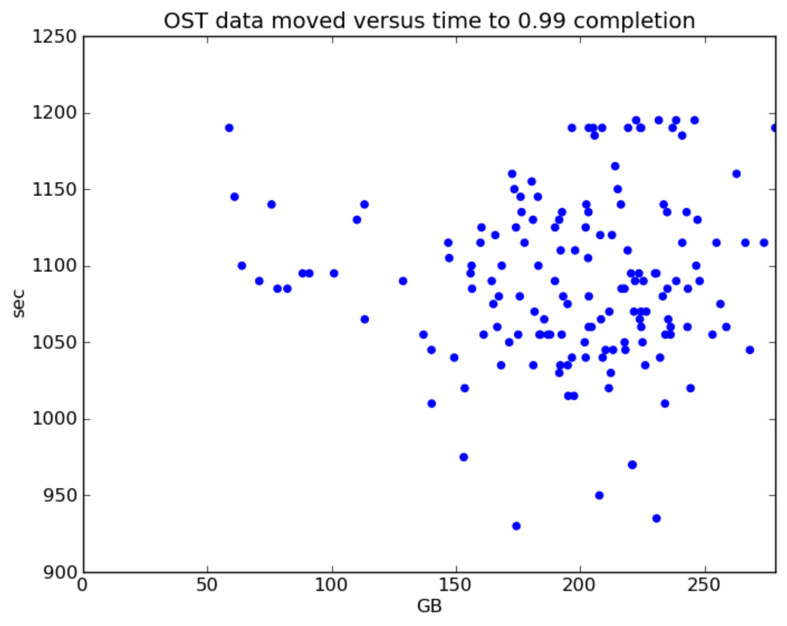
- I/O of VPIC follows a banded pattern
- Two file writing strategies
  - File per process model
  - Shared file with HDF5 and H5Part

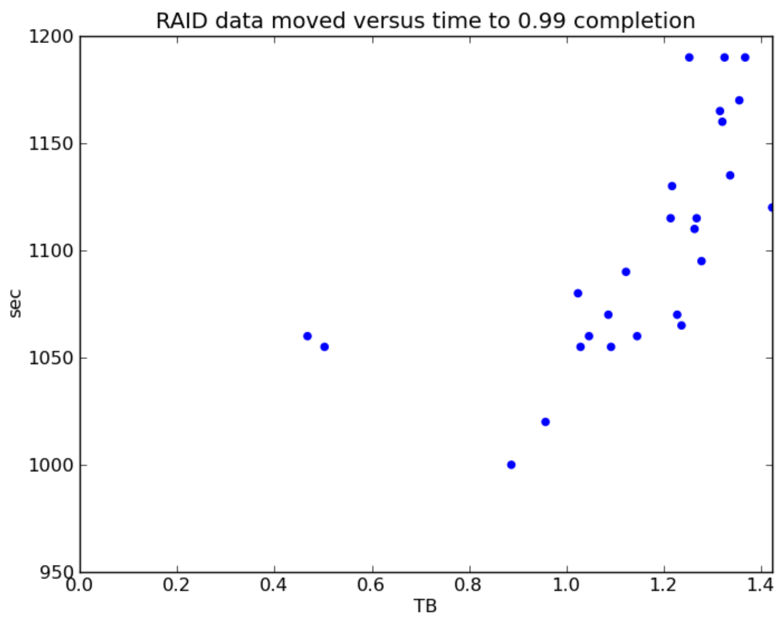
Lesson 1: Parallel HDF5 works – File-per-process

- Performance of 20,000 files with a combined size of ~30TB
- Lustre Monitoring Tool
- Load imbalance and the last OST finishing writing dictate performance
- I/O rate: 27,007 MB/s



Lesson 1: Parallel HDF5 works – File-per-process





- Uneven load leads to uneven completion times
- Problems with file per process model
  - Too many files – 20,000 files per time step in our case
  - Dictates the concurrency of subsequent stages in the analysis pipeline
  - Many data management and visualization tools only support standard data formats, such as HDF5 and NetCDF

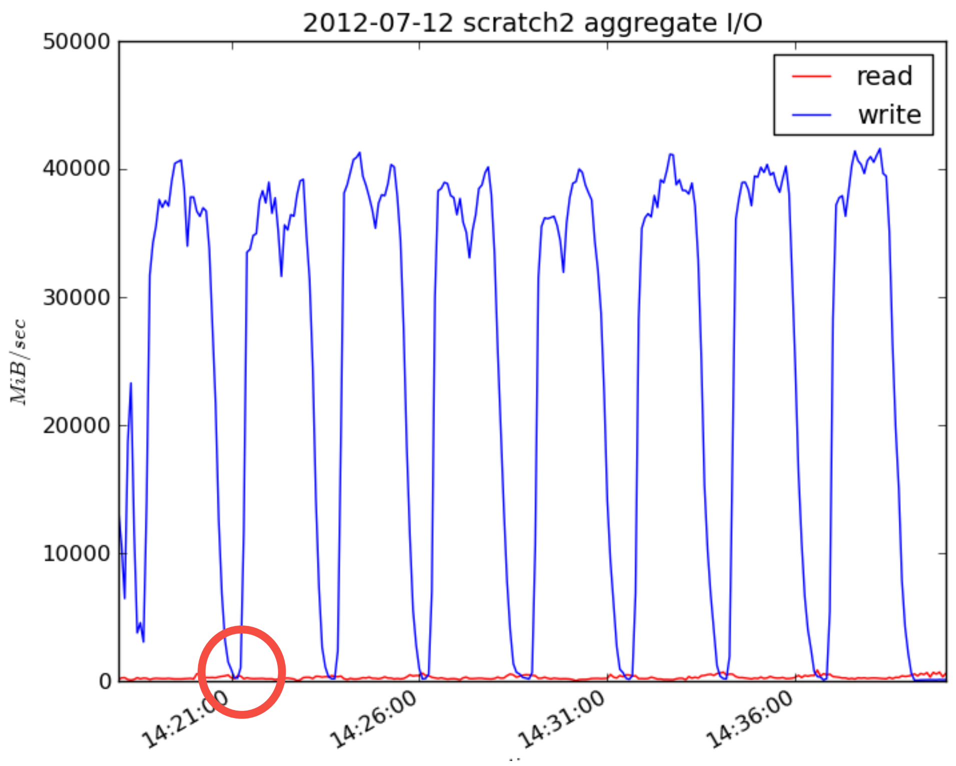
Lesson 1: Parallel HDF5 works – HDF5 and H5Part

```
h5pf = H5PartOpenFileParallel (fname, H5PART_WRITE |
                         H5PART_FS_LUSTRE, MPI_COMM_WORLD);
H5PartSetStep (h5pf, step);
H5PartSetNumParticlesStrided (h5pf, np_local, 8);

H5PartWriteDataFloat32 (h5pf, "dX", Pf);
H5PartWriteDataFloat32 (h5pf, "dY", Pf+1);
H5PartWriteDataFloat32 (h5pf, "dZ", Pf+2);
H5PartWriteDataInt32   (h5pf, "i",  Pi+3);
H5PartWriteDataFloat32 (h5pf, "Ux", Pf+4);
H5PartWriteDataFloat32 (h5pf, "Uy", Pf+5);
H5PartWriteDataFloat32 (h5pf, "Uz", Pf+6);
H5PartWriteDataFloat32 (h5pf, "q", Pf+7);

H5PartCloseFile (h5pf);
```

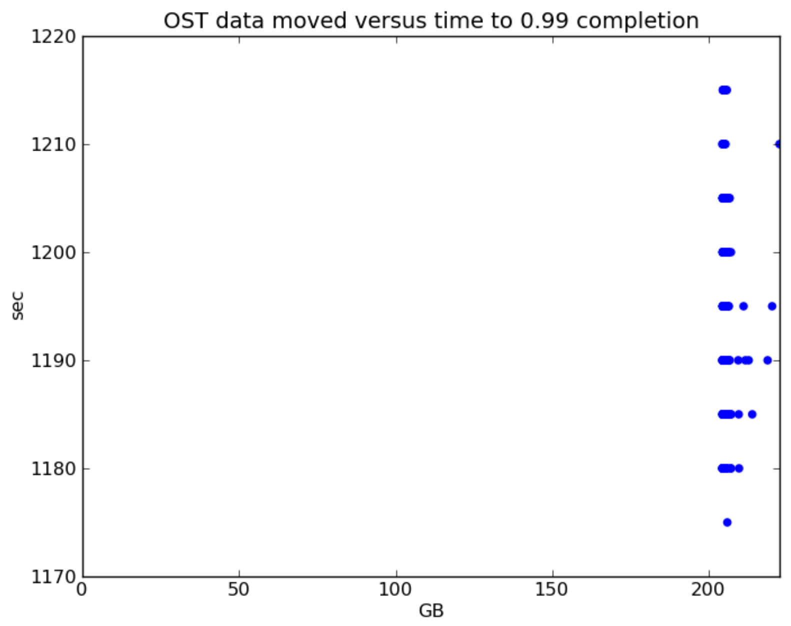
http://vis.lbl.gov/Research/H5Part/

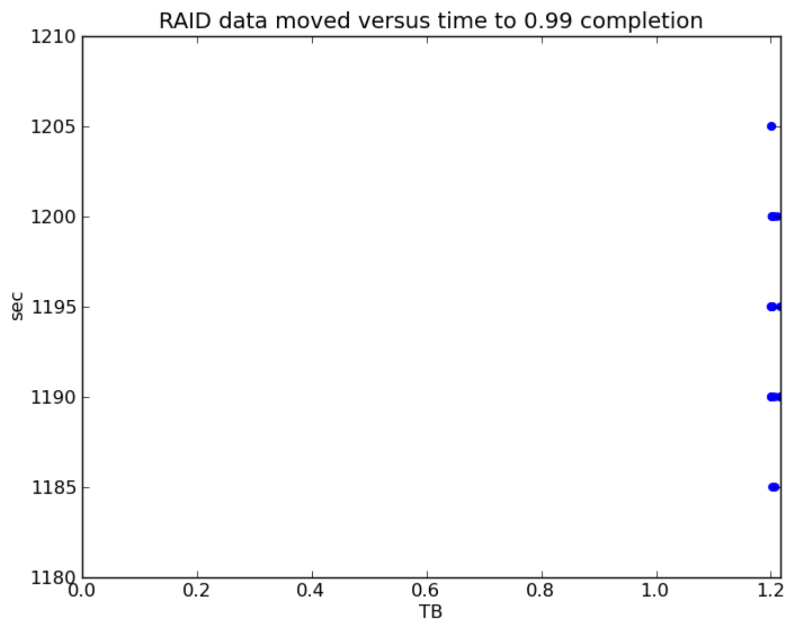
Lesson 1: Parallel HDF5 works – HDF5 and H5Part

- Performance of writing one ~31 TB particle file
- I/O rate: 27,035 MB/s
- Need for rendezvous after writing each variable, due to H5Part and HDF5 interactions



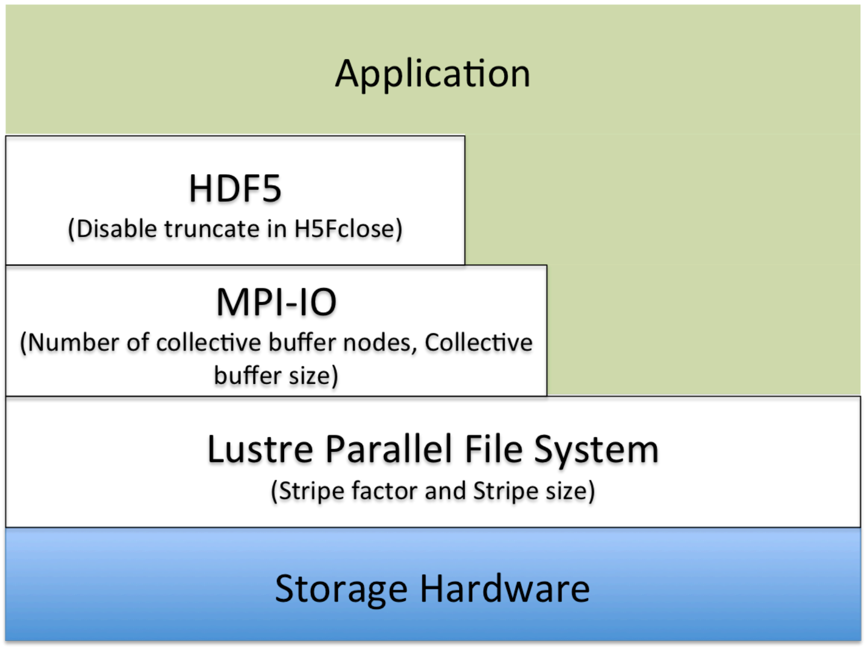
Lesson 1: Parallel HDF5 works – Load balance





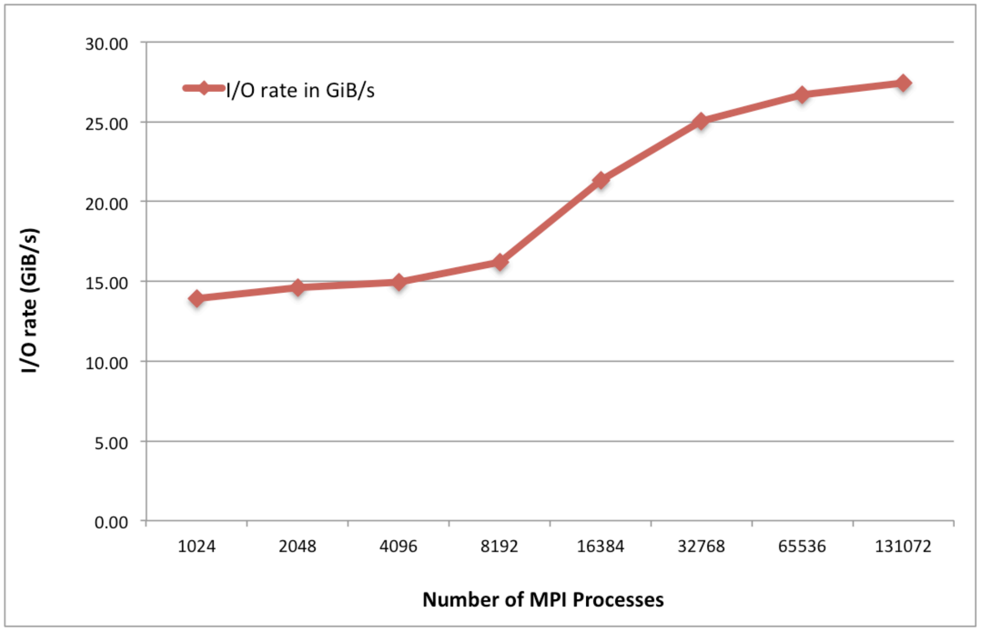
- Uniform load across the 156 OSTs and the RAIDs
- Some variability due to collective operations after each variable dump
- Overall, I/O performance of parallel HDF5 compares favorably with that of file-per-process
- HDF5 and Lustre performance was not automatic, but needed some tuning

Lesson 2: Tuning parallel I/O is a challenge, but not impossible – I/O Stack


Application
HDF5
(Disable truncate in H5Fclose)
MPI-IO
(Number of collective buffer nodes, Collective buffer size)
Lustre Parallel File System
(Stripe factor and Stripe size)
Storage Hardware

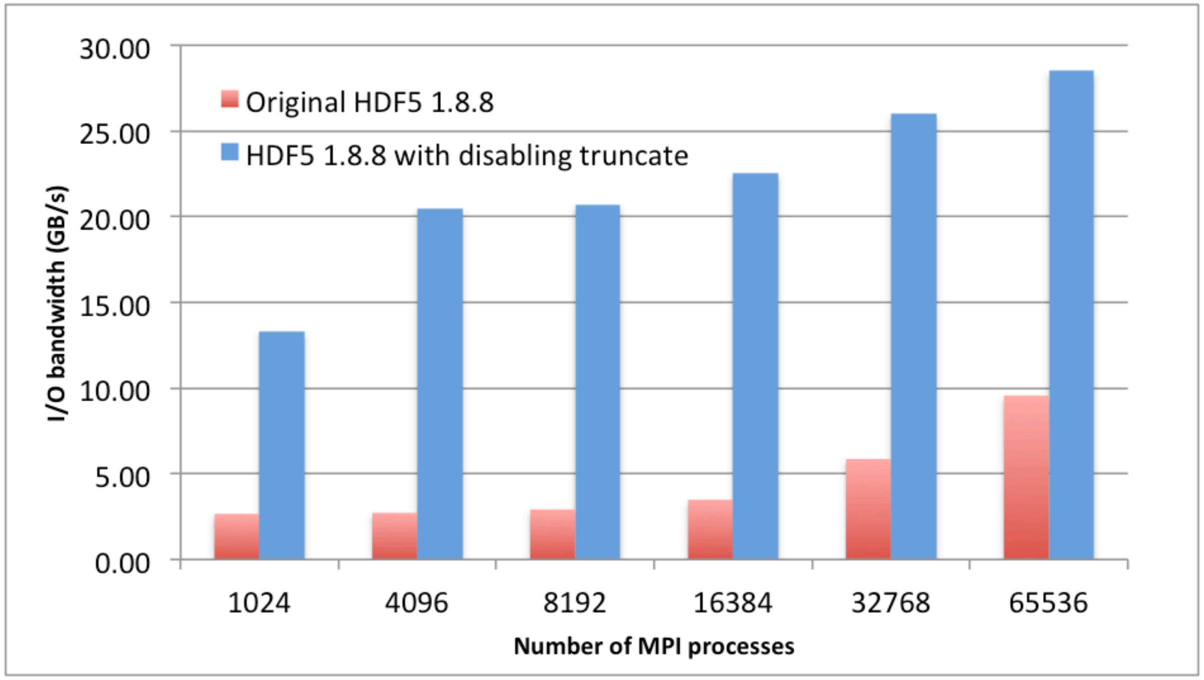
- Layers of parallel I/O software stack offer various tunable parameters
- Finding the right tunable parameters is a challenge
- To search the parameter space, we extracted the I/O kernel of VPIC
  - VPIC-IOBench
  - Two versions: Uniform and non-uniform writes from each process

Lesson 2: Tuning parallel I/O is a challenge, but not impossible – Lustre and MPI-IO tuning



- Lustre stripe count and stripe size
  - Varied stripe count from 64 to 156 and stripe size from 1MB to 1GB
  - Chose stripe count of 144 and stripe size of 64MB
- Lustre-aware MPI-IO collective buffering on Hopper uses CB2 algorithm
  - Number of collective buffering aggregator nodes is equal to the stripe count
  - Size of collective buffer is equal to the stripe size

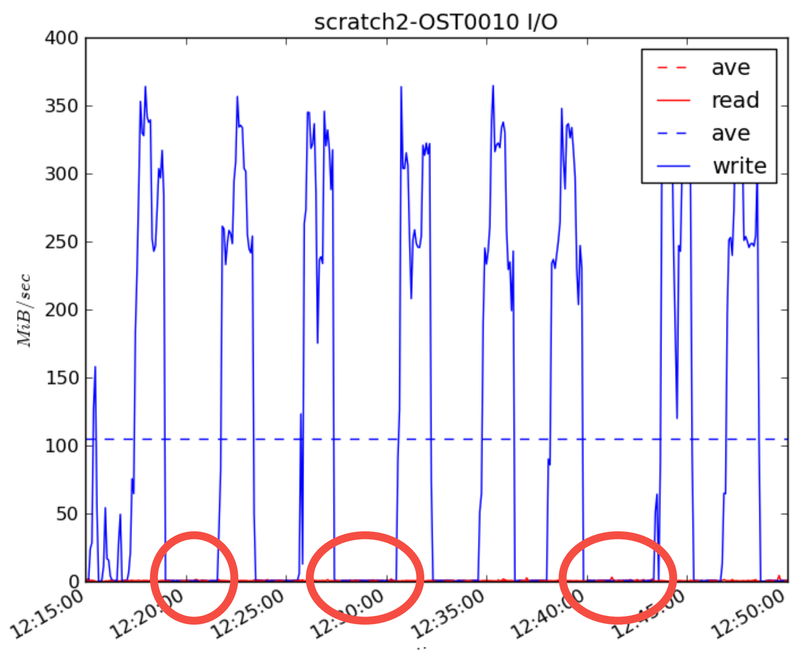
Lesson 2: Tuning parallel I/O is a challenge, but not impossible – HDF5 truncate

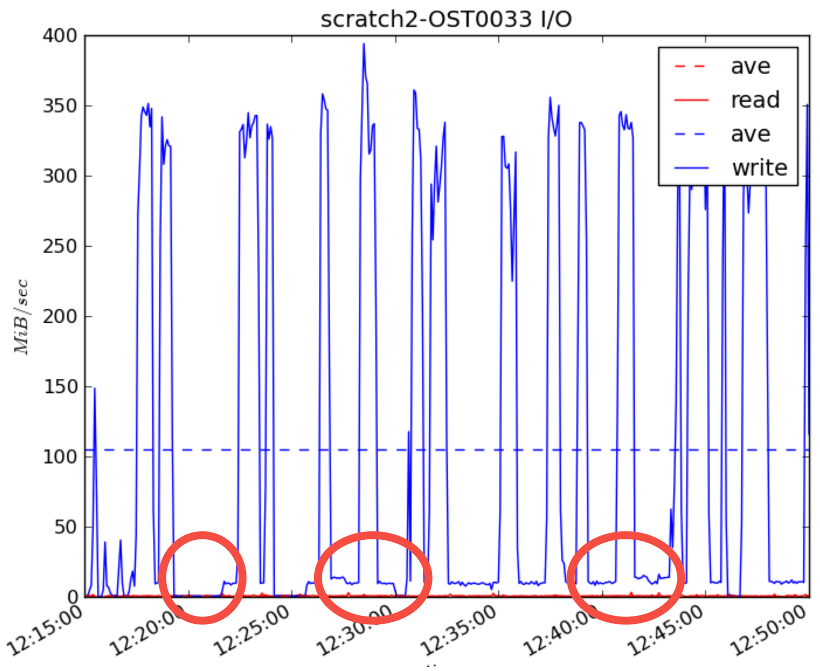


- HDF5 file close function verifies the size of the file matching with its allocated size to detect any external modification or corruption
- This is an expensive operation because of its collective nature
- Modified HDF5 to disable this "truncate" operation and achieved 3-5X performance improvement

Lesson 3: Advance verification of file system software is important – OSTs behaving badly

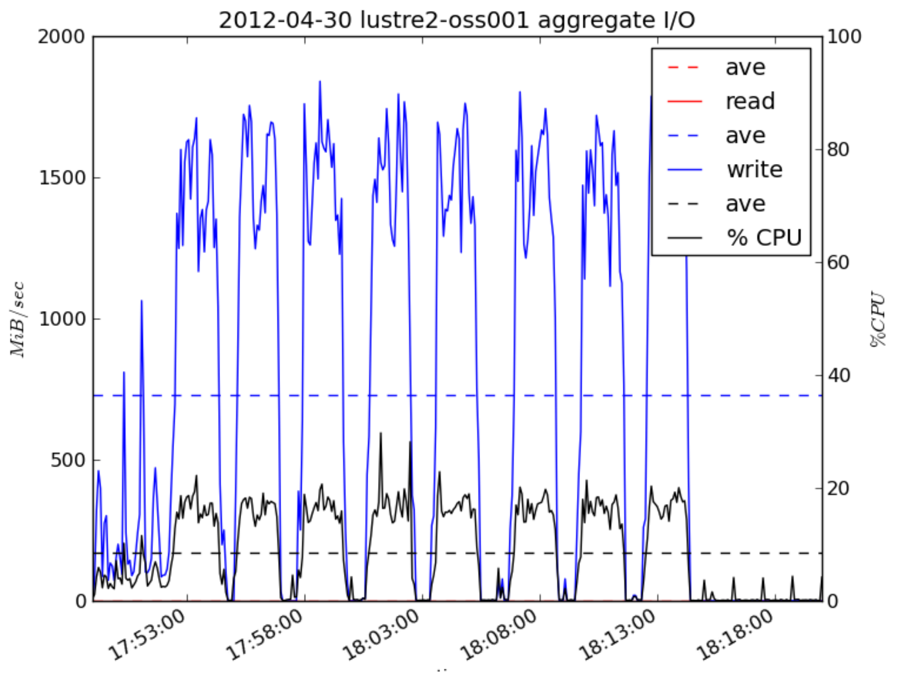
- Early runs obtained a 60% of peak bandwidth
- To achieve peak performance, each OST needs to be performing at 250 MB/s

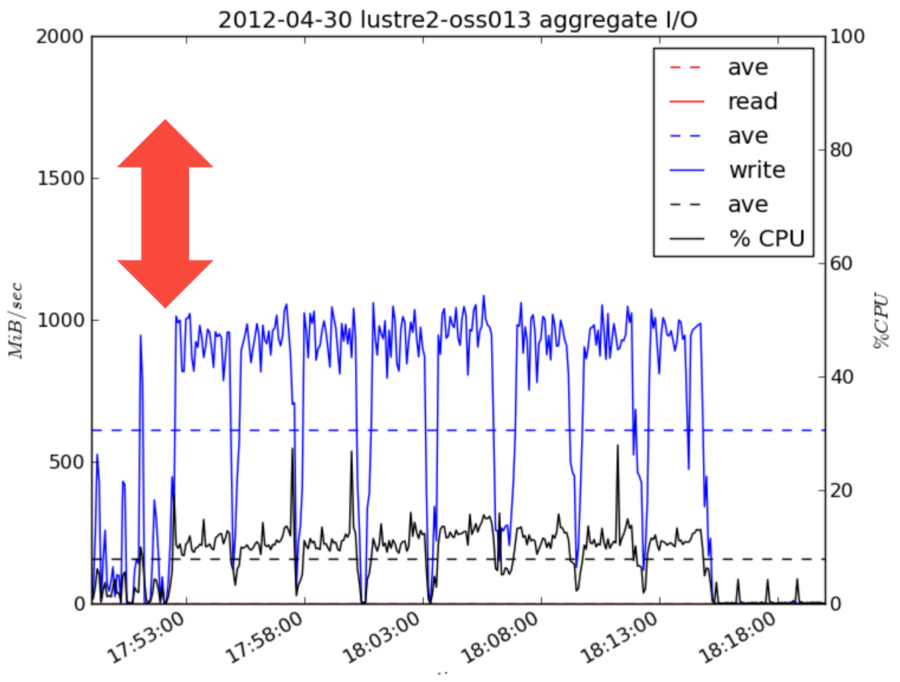




Lesson 3: Advance verification of file system software is important – OSSs behaving badly

- Early runs obtained a 60% of peak bandwidth
- To achieve peak performance, each OST needs to be performing at 250 MB/s





Lesson 4: Advance verification of resources for memory intensive apps is important

- On most nodes, Hopper has 32 GB memory
  - Some nodes have 64 GB
  - Total memory of 5,000 nodes: ~156 TB
- VPIC memory footprint is ~142 TB
  - Translates to ~29 TB on each node when the simulation uses 5,000 nodes
  - 90% of the memory on each node
- Considering some lightweight OS tasks running on the nodes, 90% of memory requirement puts significant pressure
- Experienced OOM error from one node crashing the application

Lesson 4: Advance verification of resources for memory intensive apps is important - Solution

- Used a combination of tools to verify memory availability before each run and after dumping large particle data
- Node Health Checker (NHC)
  - Free Memory Check to verify the available free memory
  - “Admindown” nodes with less than 29GB free memory
- Developed a Perl script that reads the free memory information from /proc/buddyinfo on all the nodes in allocation
  - Manually sorted and verified the free memory

Lesson 5: Scalable tools are required for diagnosing SW and HW problems

- It can be time consuming and tedious for a user to verify system health prior to a large run
- Scalable tools can help diagnosing the SW and HW problems
- Some tools exist, but need streamlining the process of verification
- Scalable computation and memory resource checker
  - With the help of Cray and NERSC staff, used NHC and “xtprocadmin” to verify the current status of nodes
  - Used NHC and local script to check memory status

Lesson 5: Scalable tools are required for diagnosing SW and HW problems

- Scalable I/O subsystem checker
  - Used manual I/O tuning to identify good set of optimization parameters
  - Our work in progress to identify tuned set of parameters at each layer of the parallel I/O stack
- Scalable Runtime I/O Monitor
  - Typically, many applications idle during I/O wasting CPU resources
  - Even one sluggish OST can increase the waste significantly
  - Lustre Monitoring Tool (LMT) was great; OSTs of bad behavior had to be found manually in postmortem – Any better and pro-active solutions??

Lessons Learned

1. Collective writes to a single shared HDF5 file can work as well as file-per-process writes
2. Tuning multiple layers of parallel I/O subsystem is a challenging task
3. Advance verification of file system hardware is important for obtaining peak performance
4. Advance verification of available resources for memory-intensive applications is important
5. Scalable tools are required for diagnosing software and hardware problems before running applications using 100k cores

Thanks!

Advanced Scientific Computing Research (ASCR) for funding the ExaHDF5 Project; Program Manager: Lucy Nowell

Application Scientists: H. Karimabadi, W. S. Daughton, V. Roytershteynz,

Colleagues: J. Chou, O. Rübel, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin, A. Shoshani, K. Wu, Q. Koziol (HDF5), J. Shalf

NERSC: Tina Butler, Katie Antypas, Francesca Verdier, Woo-Sun Yang, and Harvey Wasserman.

Cray: Steve Luzmoor, Terence Brewer, Randell Palmer, Bill Anderson, Mark Pagel, and Steven Oyanagi