I/O Best Practices

New User Training
June 16, 2020

Quincey Koziol
Data and Analytics Services Group

Outline

- **Parallel I/O**
  - **I/O Stack Overview**
  - **I/O Profiling Tools:** Darshan, Success story
  - **I/O Pattern Analysis:** Contiguous, Non-contiguous, Random, etc.
  - **I/O Libraries:** MPI-IO, HDF5, h5py
- **Burst Buffer**
  - **Architecture**
  - **Data Path:** BB to/from Lustre
  - **Success Story:** BB vs. Lustre w/ Astronomy App: H5Boss

I/O Stack: Moving Data To Disk

Productivity Interface is a thin layer on top of existing high performance I/O libraries, for productive big data analytics

Python, Spark, TensorFlow

High Level I/O Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file systems maintain a logical file model and provide efficient access to data.

Lustre, GPFS, PVFS, PanFS

Application

df=pd.read_csv("/scratch/data.csv")

Productivity Interface

High-Level I/O Library

I/O Middleware

I/O Forwarding

Parallel File System

I/O Hardware

Home: /global/homes/k/koziol
Scratch: /global/cscratch1/sd/koziol
Project: /project/projectdirs/dasrepo

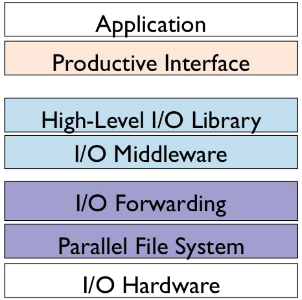
I/O Middleware organizes accesses from many processes, especially those using collective I/O.

MPI-IO, GLEAN, PLFS

I/O Forwarding transforms I/O from many clients into fewer, larger requests; reduces lock contention; and bridges between the HPC system and external storage.

IBM ciod, IOFSL, Cray DVS, Cray Datawarp

Productive I/O Interface: h5py



- Parallel h5py example:

```
from mpi4py import MPI
import h5py
fx=h5py.File(’output.h5’, ‘w’, driver=‘mpio’, comm=MPI.COMM_WORLD)
```

```
dset[start:end,:]=temp
```

Independent I/O

```
with dset.collective:
  dset[start:end,:]=temp
```

Collective I/O

NERSC BERKELEY LAB Bringing Science Solutions to the World U.S. DEPARTMENT OF ENERGY Office of Science

Coding Effort

```python
from mpi4py import MPI
import numpy as np
import h5py
import time
import sys
comm =MPI.COMM_WORLD
nproc = comm.Get_size()
comm.Barrier()
timefstart=MPI.Wtime()
f = h5py.File(filename, 'w', driver='mpio', comm=MPI.COMM_WORLD)
rank = comm.Get_rank()
dset = f.create_dataset('test', (length_x,length_y), dtype='f8')
comm.Barrier()
timefend=MPI.Wtime()
f.atomic = False
length_rank=length_x / nproc
length_last_rank=length_x -length_rank*(nproc-1)
comm.Barrier()
timestart=MPI.Wtime()
start=rank*length_rank
end=start+length_rankL
if rank==nproc-1: #last rank
    end=start+length_last_rank
temp=np.random.random((end-start,length_y))
comm.Barrier()
timemiddle=MPI.Wtime()
if colw==1:
        with dset.collective:
                dset[start:end,:] = temp
else:
        dset[start:end,:] = temp
comm.Barrier()
timeend=MPI.Wtime()
f.close()
```

```c
#include "stdlib.h"
#include "hdf5.h"
dataspace_id2 = H5Screate_simple(2, dims2, NULL);
dset_id2 = H5Dcreate(file_id2,dataset, H5T_NATIVE_DOUBLE, ..)
H5Sclose(dataspace_id2);
MPI_Barrier(comm);
double t00 = MPI_Wtime();
result_offset[1] = 0;
result_offset[0] =  (dims_x / mpi_size) * mpi_rank;
result_count[0] = dims_x / mpi_size;
result_count[1] = dims_y;
if(mpi_rank==mpi_size-1)
result_count[0] = dims_x / mpi_size + dims_x % mpi_size;
result_space = H5Dget_space(dset_id2);
H5Sselect_hyperslab(result_space, H5S_SELECT_SET, result_offset, ...);
result_memspace_size[0] = result_count[0];
result_memspace_size[1] = result_count[1];
result_memspace_id = H5Screate_simple(2, result_memspace_size, NULL);
  else{
   H5Dwrite(dset_id2, H5T_NATIVE_DOUBLE, result_memspace_id,...);
  }
  MPI_Barrier(comm);

  double t1 = MPI_Wtime()-t0;
  free(data_t);
  double tclose=MPI_Wtime();
  H5Sclose(result_space);
  H5Sclose(result_memspace_id);
  H5Dclose(dset_id2);
  H5Fclose(file_id2);
  tclose=MPI_Wtime()-tclose;
  MPI_Finalize();
}
```

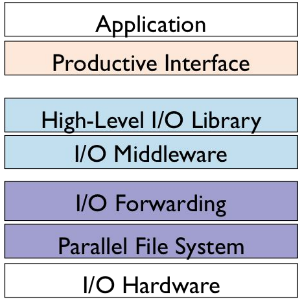
Python vs. C Performance: h5py vs. HDF5 C

Question: When you improve productivity, how much performance do you lose?

		Single Node	Multi-nodes
Metadata	1k File Creation	63.8%	
	1k Object Scanning	60.0%	
Independent I/O	Weak Scaling	97.8%	100%
	Strong Scaling	100%	97.1%
Collective I/O	Weak Scaling	100%	90%
	Strong Scaling	98.6%	87%

HDF5 vs. h5py: https://www.nersc.gov/assets/Uploads/H5py-2017-May26-public.pdf

High Level I/O Libraries



- Provide high-performance parallel I/O, while reducing complexity
  - Object-oriented data model that allows users to specify complex data relationships and dependencies
  - Have self-describing, machine-independent data formats that are suitable for array-oriented scientific data

- Examples:
  - HDF5:
    - HDF Group / LBNL, started in 1997
    - Very popular: in top 5 libraries at NERSC
  - Parallel netCDF:
    - Unidata / NWU / ANL, started in 2001
  - ADIOS:
    - ORNL / SNL, started in 2009

High-level I/O Libraries: HDF5

- Parallel HDF5 example:

```
fapl_id = H5Pcreate(H5P_FILE_ACCESS);

file_id = H5Fcreate(FNAME,…, fapl_id);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id,…);


status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, xf_id);
…
```

High-level I/O Libraries: HDF5

- Parallel HDF5 example:

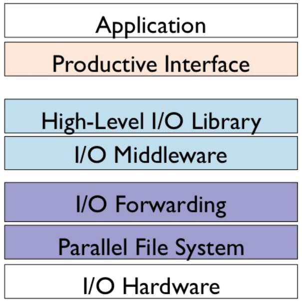
```
MPI_Init(&argc, &argv);

fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl_id, comm, info);
file_id = H5Fcreate(FNAME,…, fapl_id);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id,…);
xf_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE);
status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, xf_id);
…
MPI_Finalize();
```

High-level I/O Libraries: HDF5

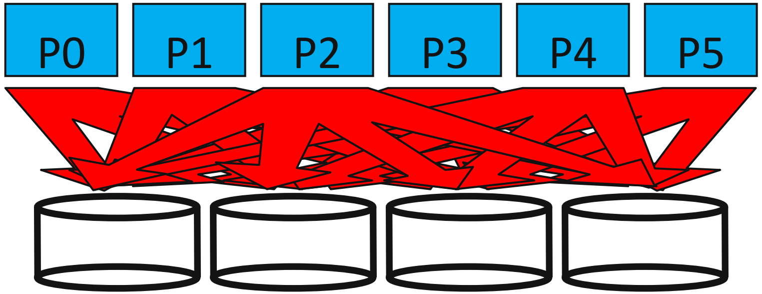
- Parallel HDF5 Tutorials:
  - NERSC:
    - https://www.nersc.gov/users/training/online-tutorials/introduction-to-scientific-i-o/
  - ATPESC:
    - https://extremecomputingtraining.anl.gov/files/2019/08/ATPESC_2019_Track-3_6_8-2_130pm_Koziol-Scalable_HDF5.pdf
  - The HDF Group:
    - https://confluence.hdfgroup.org/display/HDF5/Parallel+HDF5

I/O Middleware



- More I/O Software! Why?
  - I/O middleware provides performance portability between parallel file systems
  - Reduces or eliminates optimization in application code
- MPI-IO
  - Standardized I/O Interface specification for MPI applications
  - Same access model as POSIX: byte-stream in file
  - Features:
    - Collective I/O operations
    - Non-contiguous I/O w/MPI datatypes & file views
    - Non-blocking I/O
    - FORTRAN (and other) language bindings
    - Method of encoding files in a portable format (external32)

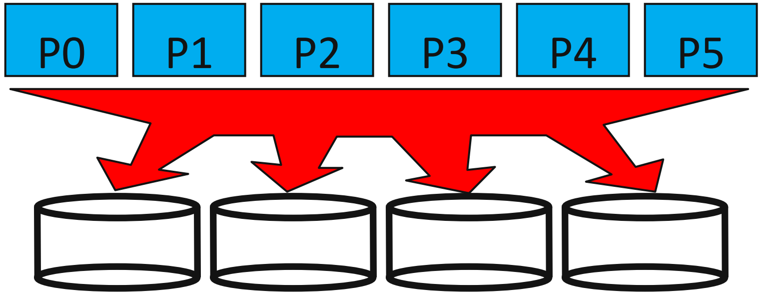
Independent and Collective Parallel I/O



Independent I/O

- Independent I/O operations specify only what a single MPI process will do
  - Independent I/O calls do no pass on relationships between I/O to other processes
- Why use independent I/O?
  - Sometimes the synchronization of collective calls is not natural
  - Sometimes the overhead of collective calls outweighs their benefits
    - Example: Very small metadata I/O operations

Independent and Collective Parallel I/O



Collective I/O

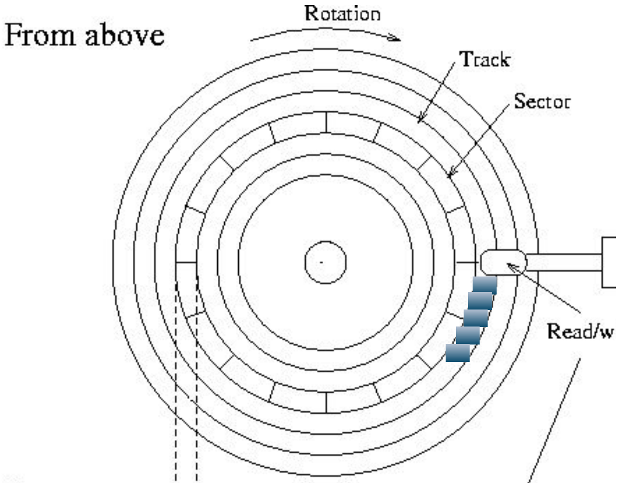
- Collective I/O operations are coordinated access by a group of MPI processes
  - Collective I/O routines must be called by all processes that opened the file
- Why use collective I/O?
  - Allows I/O middleware to get a global perspective on entire access from all processes, providing more opportunities for optimization in lower software layers
  - When used for non-contiguous access patterns, collective I/O typically yields the best performance

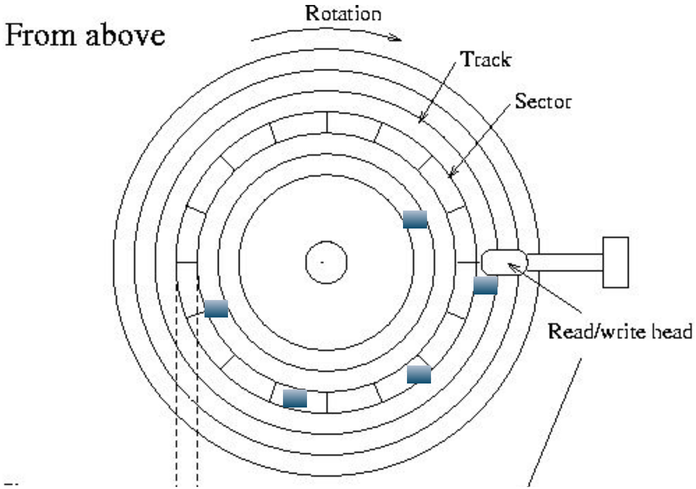
I/O Middleware

- MPI-IO Tutorials:
  - NERSC:
    - https://docs.nersc.gov/performance/io/library/#mpi-io-tuning
  - ATPESC:
    - https://extremecomputingtraining.anl.gov/files/2019/08/ATPESC_2019_Track-3_4_8-2_1030am_Latham-Introduction_to_MPI_IO.pdf
  - NCSA:
    - http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture32.pdf

I/O Pattern Analysis

How to describe your I/O
- Number of Processes
- Number of Files
- Size per file
- Frequency of I/O
- Size per I/O
- Read or Write or ?
- Shared File or not
- I/O Libraries
- ...





What is your I/O Pattern?
- Contiguous or Non-contiguous?
- (i.e. Sequential or Random?)

Contiguous I/O
- read time, 0.1ms

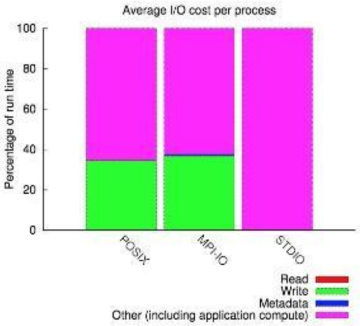
Noncontiguous I/O
- Seek time, 4ms
- Rotation time, 3ms
- Read time, 0.1 ms
- Total time: 7.1ms

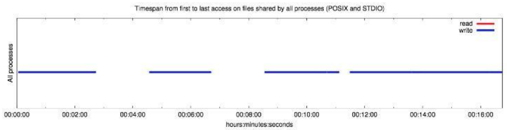
I/O Profiling

- Darshan
  - Lightweight HDF5/MPI-IO/POSIX I/O profiling tool, developed by ANL
  - Loaded by default for all NERSC users: “module load darshan”
    - module list: darshan/3.1.7
- Darshan Log
  - Location: /global/cscratch1/sd/darshanlogs/<year>/<month>/<day>/
    - Around 5000 logs / day
  - Filename format: Username_Jobname_SlurmId_JobId_xxx.darshan
    - Example: zisheng_vasp_gam_id31418939_6-11-85148-14130502361054725666_1.darshan
- Darshan Scripts:
  - “darshan-job-summary.pl <xxx>.darshan”
  - “darshan-summary-per-file.sh <xxx>.darshan”

I/O Profiling Success Story: Athena's I/O

Athena is an astrophysics code, used in wide range of problems: interstellar medium, star formation, etc.





"I made the changes you suggested and did the test. It solved my problem! Previously, **the I/O can take 40% of the time. Now the I/O time is basically 0**.

Thank you very much for your help. This is really useful."

---Dr. Yan-Fei Jiang, Harvard

IO Analysis with Darshan: "darshan-job-summary.pl <darshan_log> <output.pdf>"

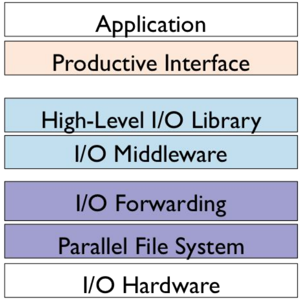
Darshan: https://docs.nersc.gov/performance/io/ And https://www.mcs.anl.gov/research/projects/darshan/
HDF5: https://docs.nersc.gov/development/libraries/hdf5/ And https://www.hdfgroup.org/solutions/hdf5/
Athena: https://princetonuniversity.github.io/athena/

NERSC

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

NERSC Storage Systems



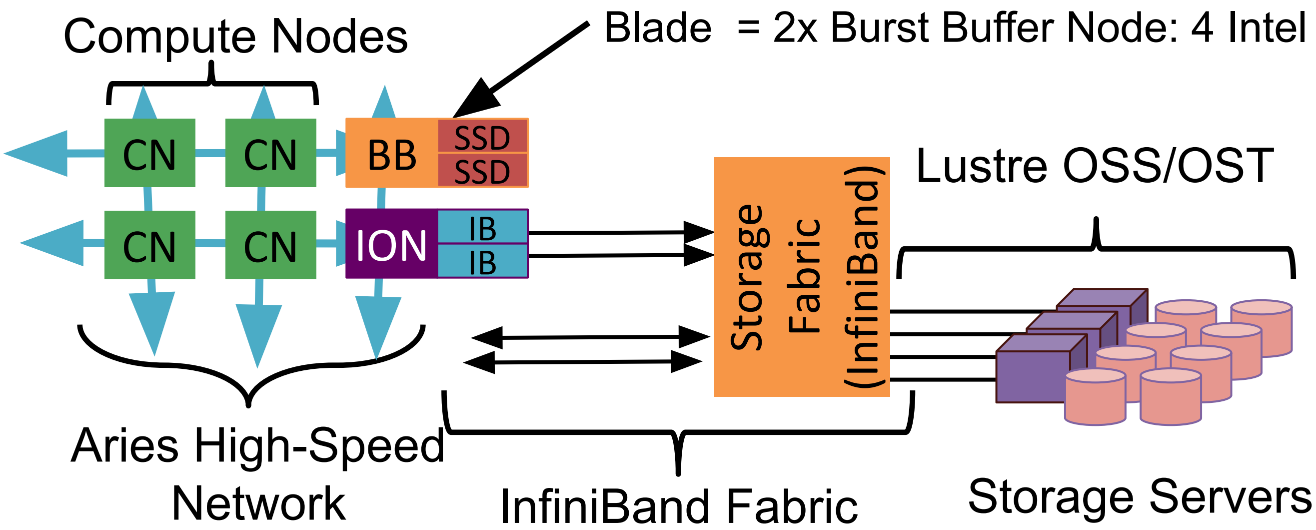
NERSC File Systems: https://docs.nersc.gov/filesystems/
https://docs.nersc.gov/performance/io/

Location	File System	Visibility	Access	Backups?	Snapshots?	Purged?
Home	GPFS	Global	User	Yes	Yes	No
Common	GPFS	Global	Repository	No	No	No
Community	GPFS	Global	Repository	No	Yes	No
Scratch	Lustre	Local	User	No	No	Yes
Burst Buffer	DataWarp	Local	Job	No	No	Yes
Archive	HPSS	Global	User	No	No	No

Cori Scratch: Lustre Overview

- Lustre is a high-performance parallel file system
  - POSIX File System
    - Directories & Files
  - Each file's data can be striped over multiple storage servers ("OSTs")
    - Default is a "stripe count" of 1, i.e. not striped
    - "Stripe size" is amount of data in each stripe, with data "round robined" over # of OSTs for file
  - Files inherit striping configuration of directory where they are created
  - More details at: https://docs.nersc.gov/performance/io/lustre/

Burst Buffer: Architecture

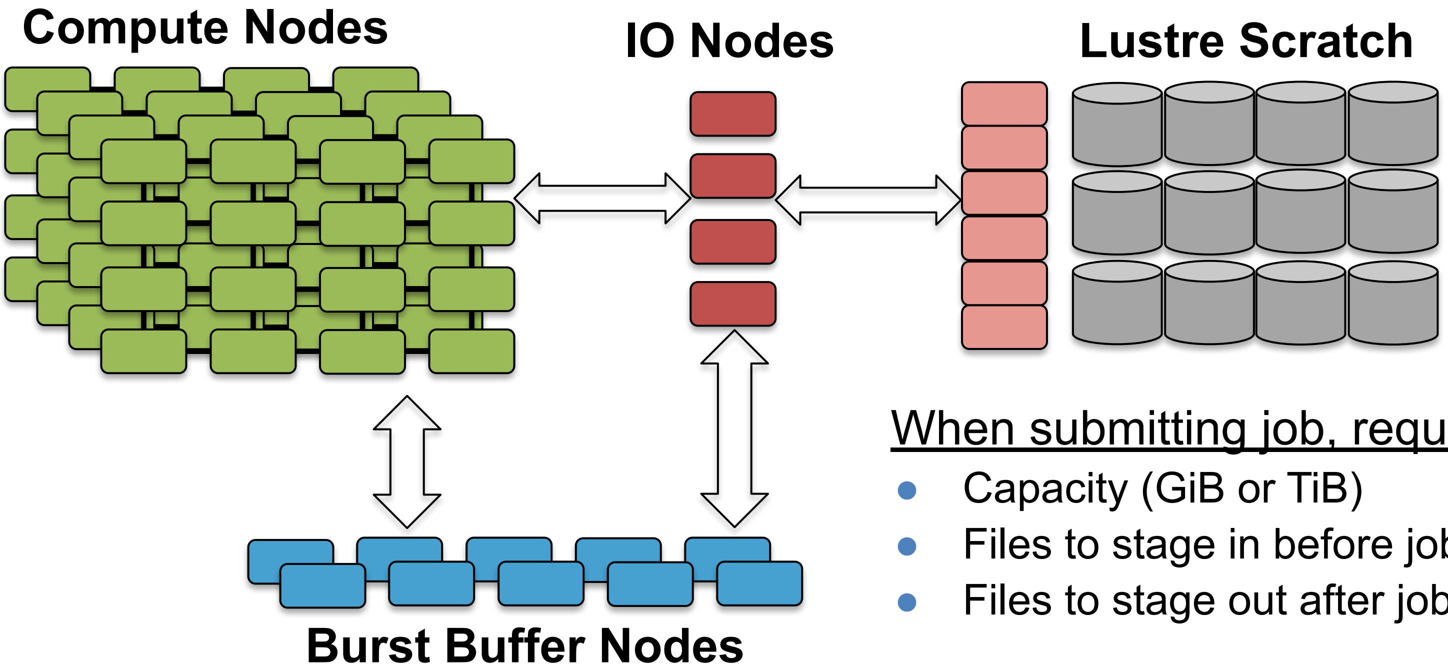


	Burst Buffer	Lustre
Nodes	288	248
Capacity (PB)	1.8	28

- DataWarp software (integrated with SLURM WLM) allocates portions of available Burst Buffer storage to users either per-job or 'persistent'.
- Users see Burst Buffer as a POSIX filesystem
- Filesystem can be striped across multiple Burst Buffer nodes (depending on allocation size requested)

Burst Buffer: https://docs.nersc.gov/filesystems/cori-burst-buffer/
https://docs.nersc.gov/performance/io/bb/

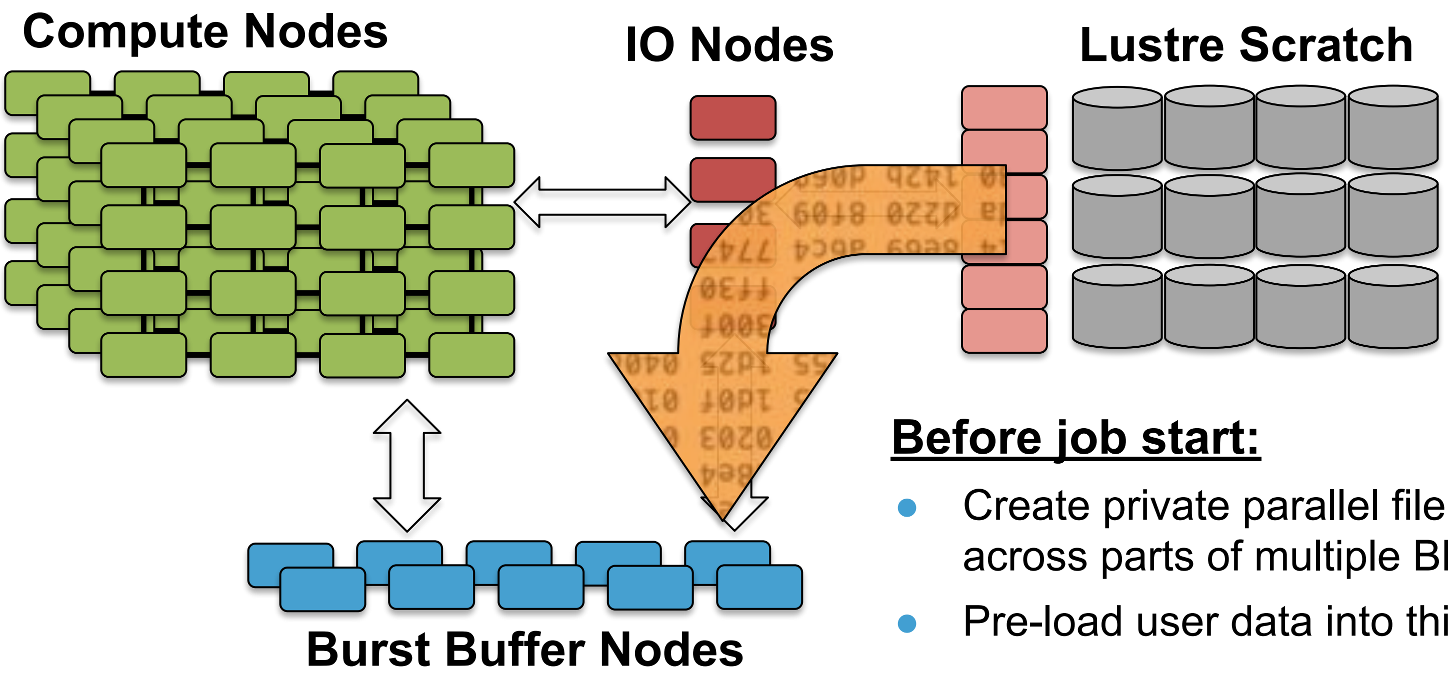
Burst Buffer: Data Paths



<u>When submitting job, request:</u>

- Capacity (GiB or TiB)
- Files to stage in before job starts
- Files to stage out after job finishes

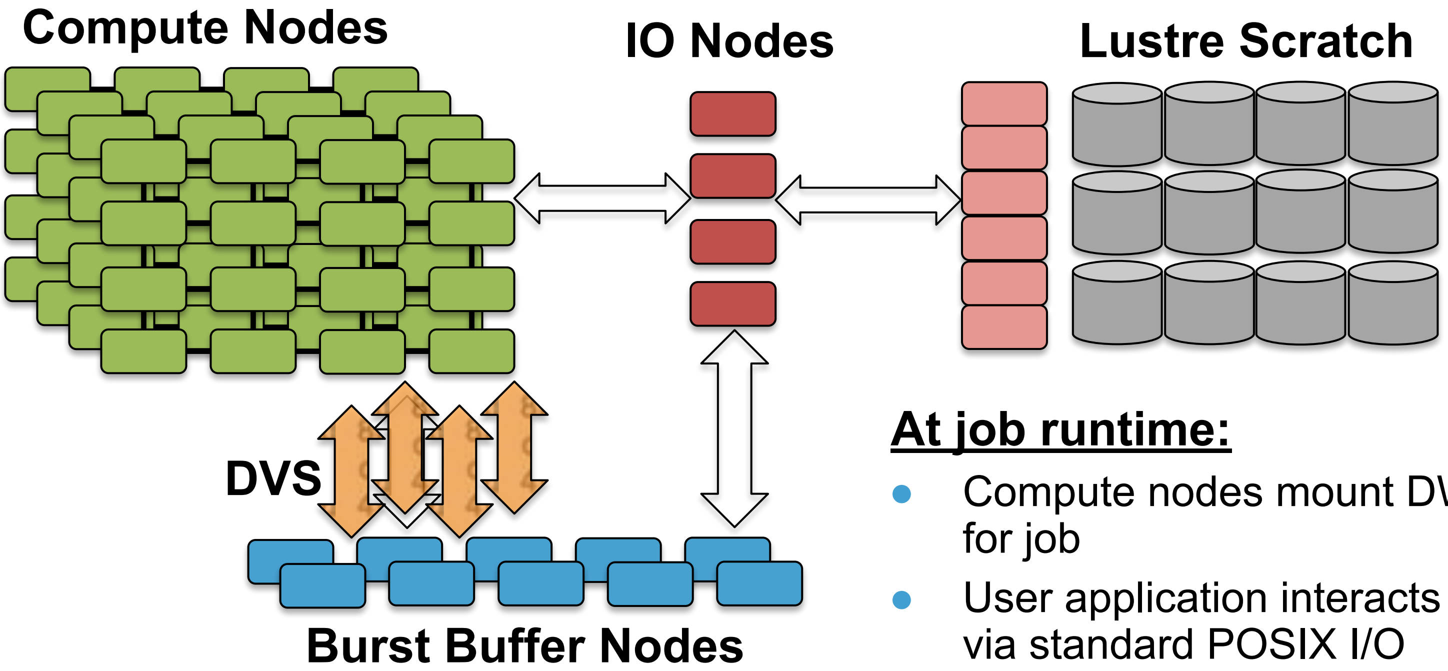
Burst Buffer: Data Paths



Before job start:

- Create private parallel file system (DWFS) across parts of multiple BB nodes
- Pre-load user data into this DWFS

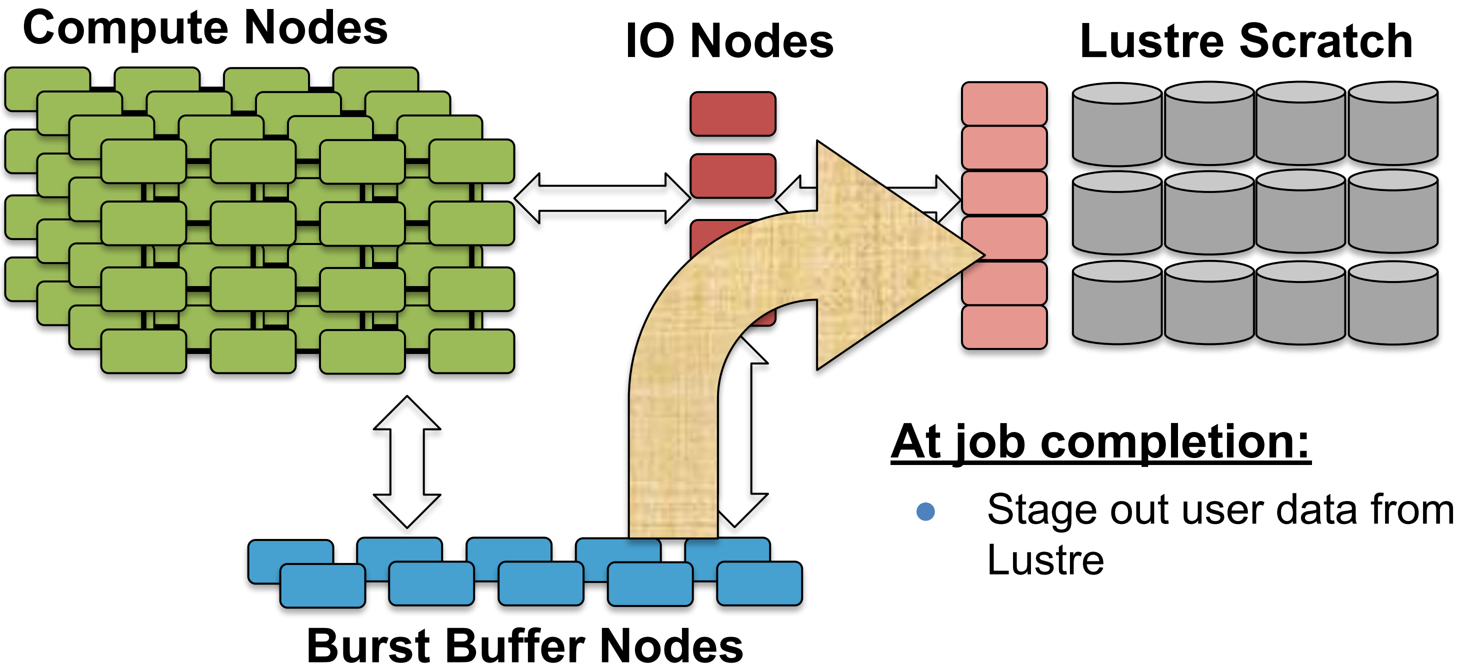
Burst Buffer: Data Paths



At job runtime:

- Compute nodes mount DWFS created for job
- User application interacts with DWFS via standard POSIX I/O

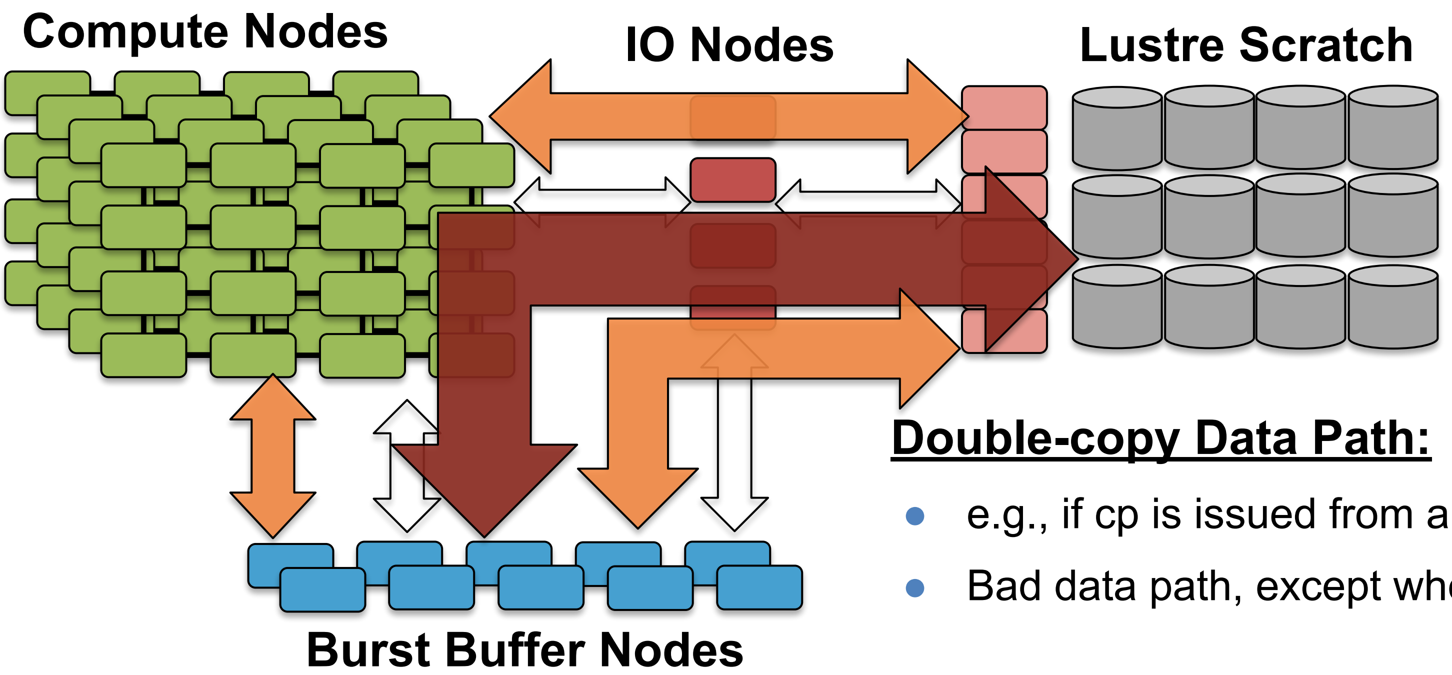
Burst Buffer: Data Paths



At job completion:

- Stage out user data from DWFS to Lustre

Burst Buffer: Data Paths

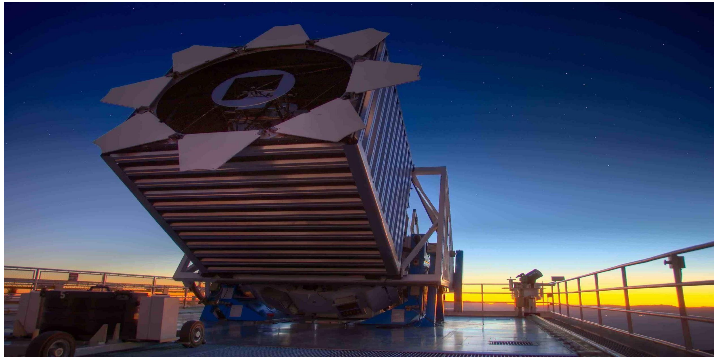


Double-copy Data Path:

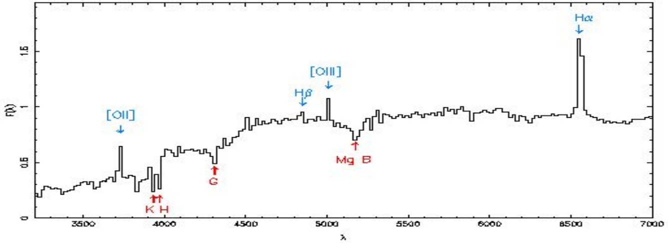
- e.g., if cp is issued from a compute node
- Bad data path, except when #CN >> #BBNs

Burst Buffer: Success Story – H5Boss

- Selecting subsets of galaxy spectra from a large dataset
  - Small, random memory accesses
  - Typical web query for SDSS dataset



Time taken to extract 1000 random spectra	From one HDF5 file	From one FITS file
From Lustre	44.1s	160.3s
From BB	1.3s	44.0s
Speedup:	**33x**	3.6x



Thank You and Welcome to NERSC!