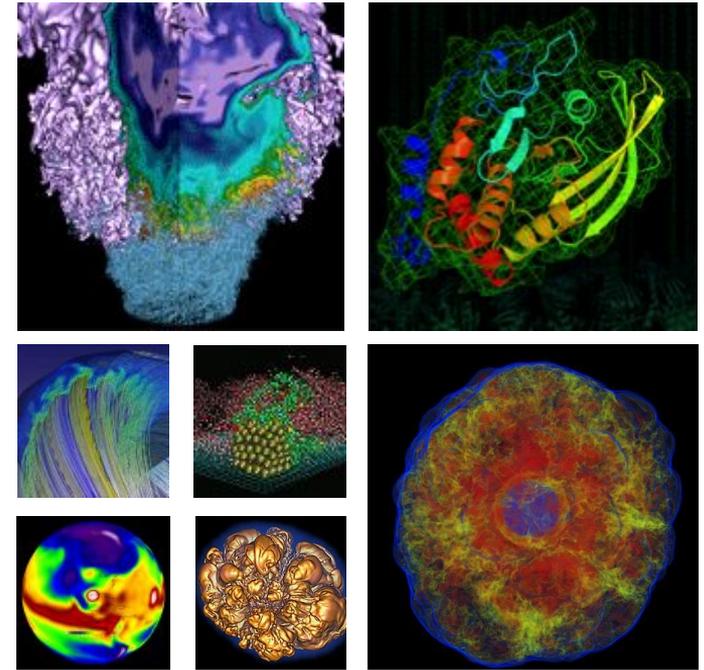


I/O Best Practices



June 21st, 2019

New User Training

Quincey Koziol
Data Analytics Service
Group

- **Parallel I/O**
 - **I/O Stack Overview**
 - **I/O Profiling Tools:** Darshan, Successful 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; Astronomy Apps: H5Boss

I/O Stack: Moving Data To Disk



Productive Interface builds a thin layer on top of existing high performance I/O library for productive big data analytics

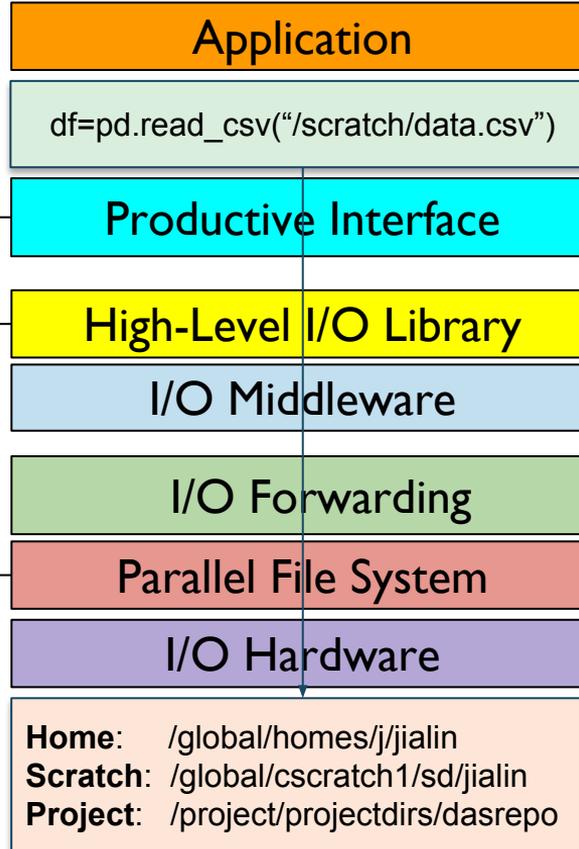
Python, Spark, TensorFlow

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

HDF5, Parallel netCDF, ADIOS

Parallel file system maintains logical file model and provides efficient access to data.

PVFS, PanFS, GPFS, Lustre



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 request; 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

```
1 from mpi4py import MPI
2 import h5py
3 fx=h5py.File('output.h5', 'w', driver='mpio', comm=MPI.COMM_WORLD)
```

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

Independent I/O

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

Collective I/O

Coding Effort



```
1 from mpi4py import MPI
2 import numpy as np
3 import h5py
4 import time
5 import sys
6 comm = MPI.COMM_WORLD
7 nproc = comm.Get_size()
8 comm.Barrier()
9 timefstart=MPI.Wtime()
10 f = h5py.File(filename, 'w', driver='mpio', comm=MPI.COMM_WORLD)
11 rank = comm.Get_rank()
12 dset = f.create_dataset('test', (length_x,length_y), dtype='f8')
13 comm.Barrier()
14 timefend=MPI.Wtime()
15 f.atomic = False
16 length_rank=length_x / nproc
17 length_last_rank=length_x -length_rank*(nproc-1)
18 comm.Barrier()
19 timestart=MPI.Wtime()
20 start=rank*length_rank
21 end=start+length_rankL
22 if rank==nproc-1: #last rank
23     end=start+length_last_rank
24 temp=np.random.random((end-start,length_y))
25 comm.Barrier()
26 timemiddle=MPI.Wtime()
27 if colw==1:
28     with dset.collective:
29         dset[start:end,:] = temp
30 else:
31     dset[start:end,:] = temp
32 comm.Barrier()
33 timeend=MPI.Wtime()
34 f.close()
```



```
1 #include "stdlib.h"
2 #include "hdf5.h"
35 dataspace_id2 = H5Screate_simple(2, dims2, NULL);
36 dset_id2 = H5Dcreate(file_id2,dataset, H5T_NATIVE_DOUBLE,
37 H5Screate_simple(2, dims2, NULL),
38 MPI_Barrier(comm);
39 double t00 = MPI_Wtime();
40 result_offset[1] = 0;
41 result_offset[0] = (dims_x / mpi_size) * mpi_rank;
42 result_count[0] = dims_x / mpi_size;
43 result_count[1] = dims_y;
44 if(mpi_rank==mpi_size-1)
45 result_count[0] = dims_x / mpi_size + dims_x % mpi_size;
46 result_space = H5Dget_space(dset_id2);
47 H5Sselect_hyperslab(result_space, H5S_SELECT_SET, result_offset, ...);
48 result_memspace_size[0] = result_count[0];
49 result_memspace_size[1] = result_count[1];
50 result_memspace_id = H5Screate_simple(2, result_memspace_size, NULL);
68 else{
69     H5Dwrite(dset_id2, H5T_NATIVE_DOUBLE, result_memspace_id,...);
70 }
71 MPI_Barrier(comm);
72
73 double t1 = MPI_Wtime()-t0;
74 free(data_t);
75 double tclose=MPI_Wtime();
76 H5Sclose(result_space);
77 H5Sclose(result_memspace_id);
78 H5Dclose(dset_id2);
79 H5Fclose(file_id2);
80 tclose=MPI_Wtime()-tclose;
81 MPI_Finalize();
82 }
```



h5py vs. HDF5 Performance



h5Py Performance / HDF5 Performance

Questions: When you gain the productivity, how much performance you can afford to 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%

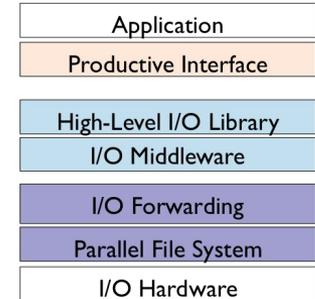
High Level I/O Libraries



- Take advantage of high-performance parallel I/O while reducing complexity
 - Add a well-defined layer to the I/O stack
 - Allow users to specify complex data relationships and dependencies
 - Come with machine-independent data formats, self-describing, suitable for array-oriented scientific data

- Examples

- HDF5: HDF group, since 1989, top 5 libraries at NERSC
- Parallel netCDF: NWU, ANL, since 2001
- ADIOS: ORNL, since 2009



High Level I/O Libraries: HDF5



- A parallel HDF5 program has a few extra calls over a serial one

```
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,...);
```

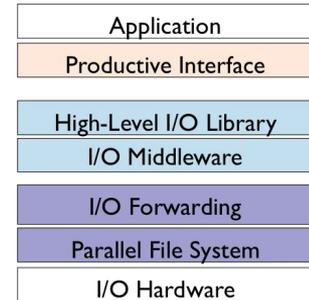
```
xf_id = H5Pcreate(H5P_DATASET_XFER);
```

```
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE);
```

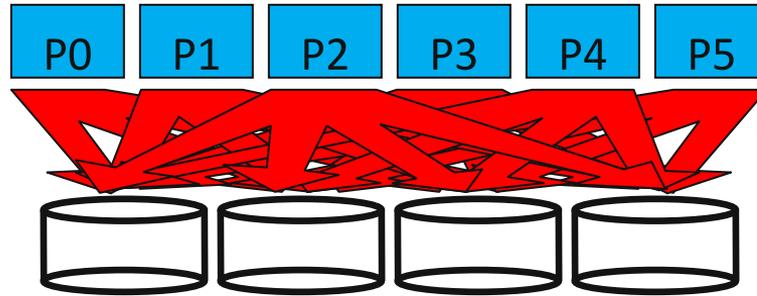
```
status = H5Dwrite(dset_id, H5T_NATIVE_INT, ..., xf_id);
```

```
MPI_Finalize();
```

- Why additional I/O Software?
 - Additional I/O software provides improved performance and usability over directly accessing the parallel file system.
 - Reduces or eliminates need for optimization in application codes.
- MPI-IO
 - I/O interface specification for use in MPI apps
 - Data model is same as POSIX: Stream of bytes in a file
- MPI-IO Features
 - Collective I/O
 - Noncontiguous I/O with MPI datatypes and file views
 - Nonblocking I/O
 - FORTRAN bindings (and additional languages)
 - Method of encoding files in a portable format (external32)



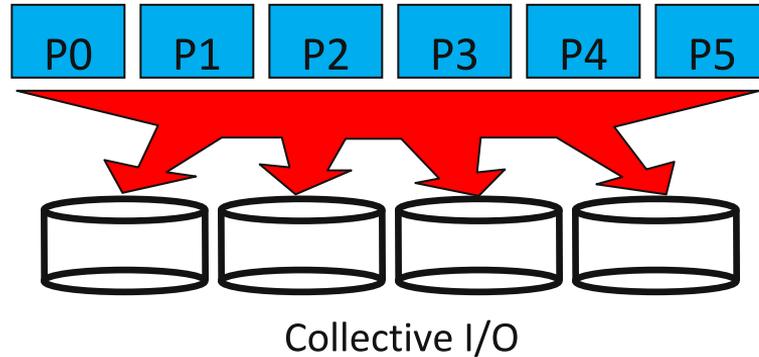
Independent and Collective I/O



Independent I/O

- Independent I/O operations specify only what a single process will do
 - Independent I/O calls do not pass on relationships between I/O on 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 I/O during metadata operations

Independent and Collective I/O



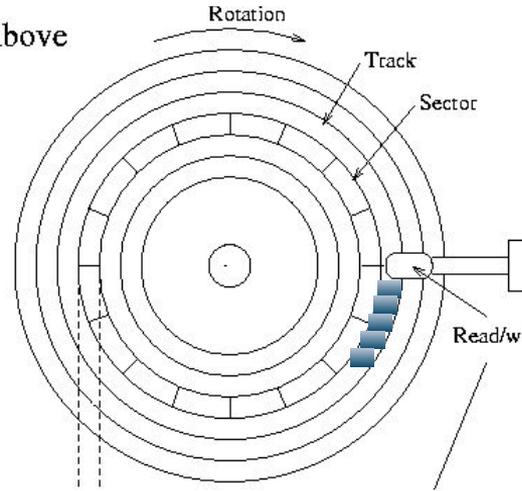
- Collective I/O is coordinated access to storage by a group of processes
 - Collective I/O functions are called by all processes participating in I/O
- Why use collective I/O
 - Allows I/O layers to know more about access as a whole, more opportunities for optimization in lower software layers, better performance
 - Combined with non-contiguous accesses yields highest performance

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
- ...

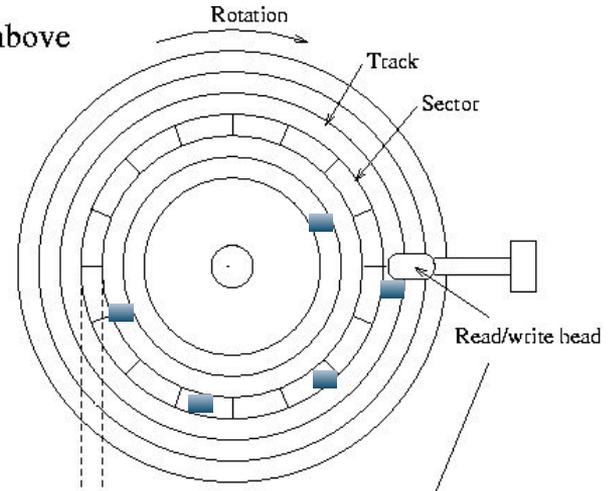
From above



Contiguous I/O

- read time, **0.1ms**

From above



Noncontiguous I/O

- Seek time, 4ms
- Rotation time, 3ms
- Read time, 0.1 ms
- Total time: **7.1ms**

What is your I/O Pattern?

- Contiguous or Non-contiguous?
- (i.e. Sequential or Random?)

Darshan

- Loaded by default for all NERSC users: “module load darshan”
- module list: darshan/3.1.4
- MPI-IO/POSIX/HDF5 I/O lightweight profiling tool, developed by ANL

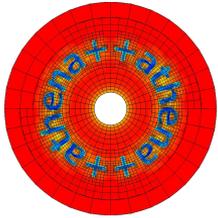
Darshan Log

- Location: /global/cscratch1/sd/darshanlogs/2019/6/21/
- Statistics: 5000~ logs per day
- Format: `Username_Jobname_idSlurm_JobId_xxx.darshan`
- Example: `zulissi_vasp_std_id16178922_11-4-71121-6884128420676186490_3.darshan`

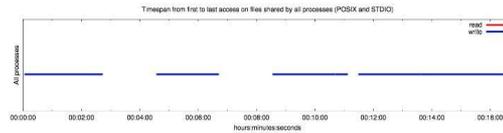
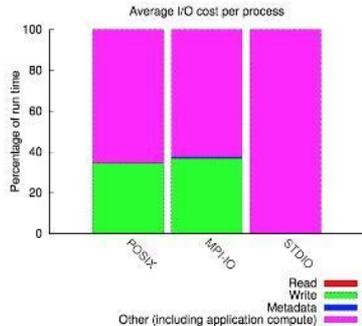
Darshan Command

- `darshan-job-summary.pl` `xxxx.darshan`
- `darshan-summary-per-file.sh` `xxxx.darshan`

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.”

IO Analysis with Darshan

---Dr. Yan-Fei Jiang, Harvard

darshan-job-summary.pl darshan_log output.pdf

Darshan: <http://www.nersc.gov/users/software/performance-and-debugging-tools/darshan/>

HDF5: <http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/>

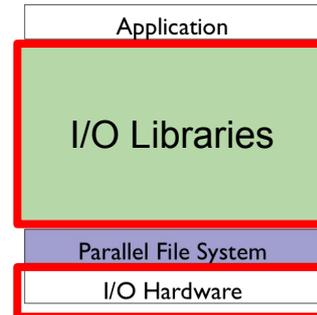
Athena: <https://princetonuniversity.github.io/athena/>

- Parallel I/O
 - I/O Stack Overview
 - I/O Libraries: MPI-IO, HDF5, H5Py
 - I/O Pattern Analysis: Contiguous, Non-contiguous, Random, etc
 - I/O Profiling Tools: Darshan, Success Story

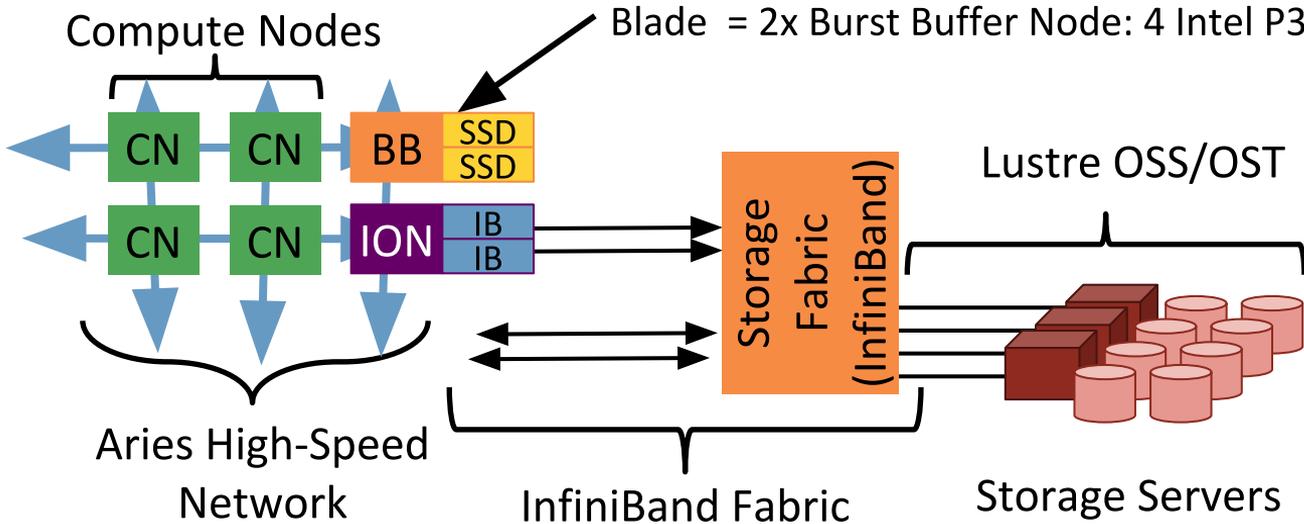
File system is also important: Lustre / GPFS / HPSS/ DataWarp

<http://www.nersc.gov/users/storage-and-file-systems/i-o-resources-for-scientific-applications/optimizing-io-performance-for-lustre/>

- **Burst Buffer: ~~Hard Disk Drive~~ Solid State Drive**



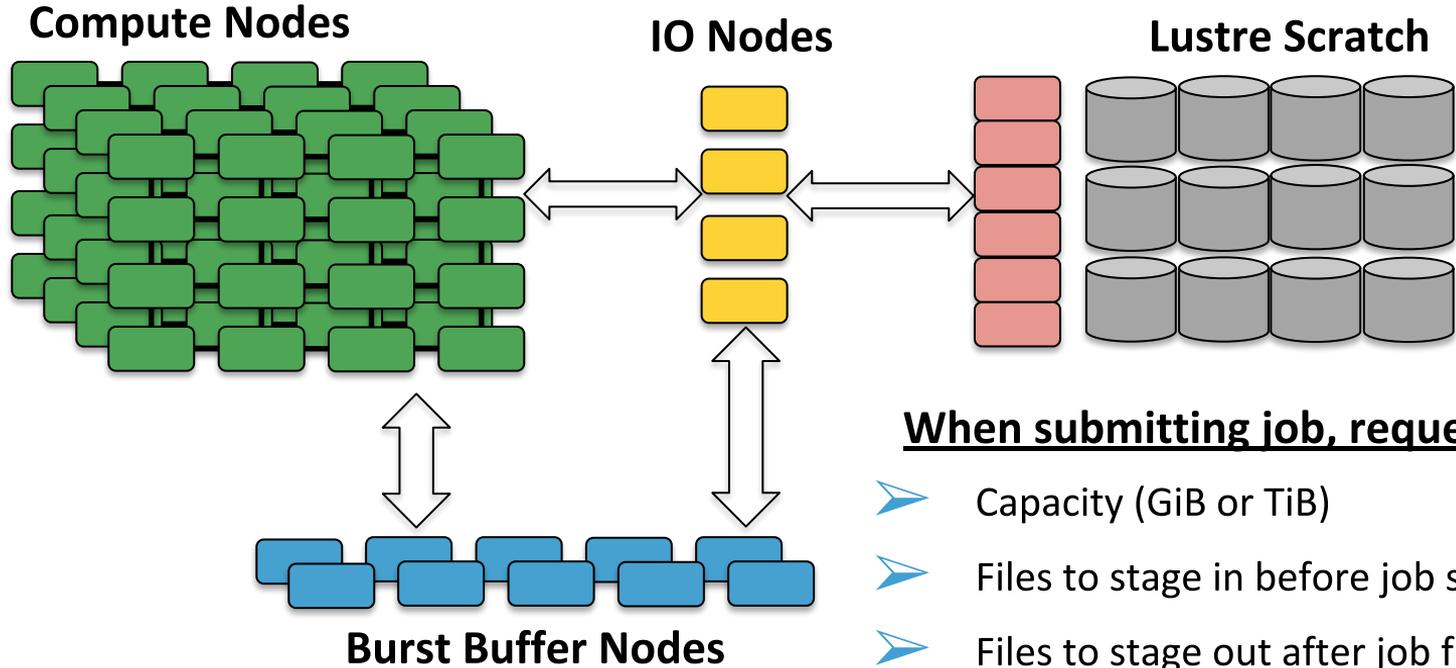
Burst Buffer Architecture



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

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

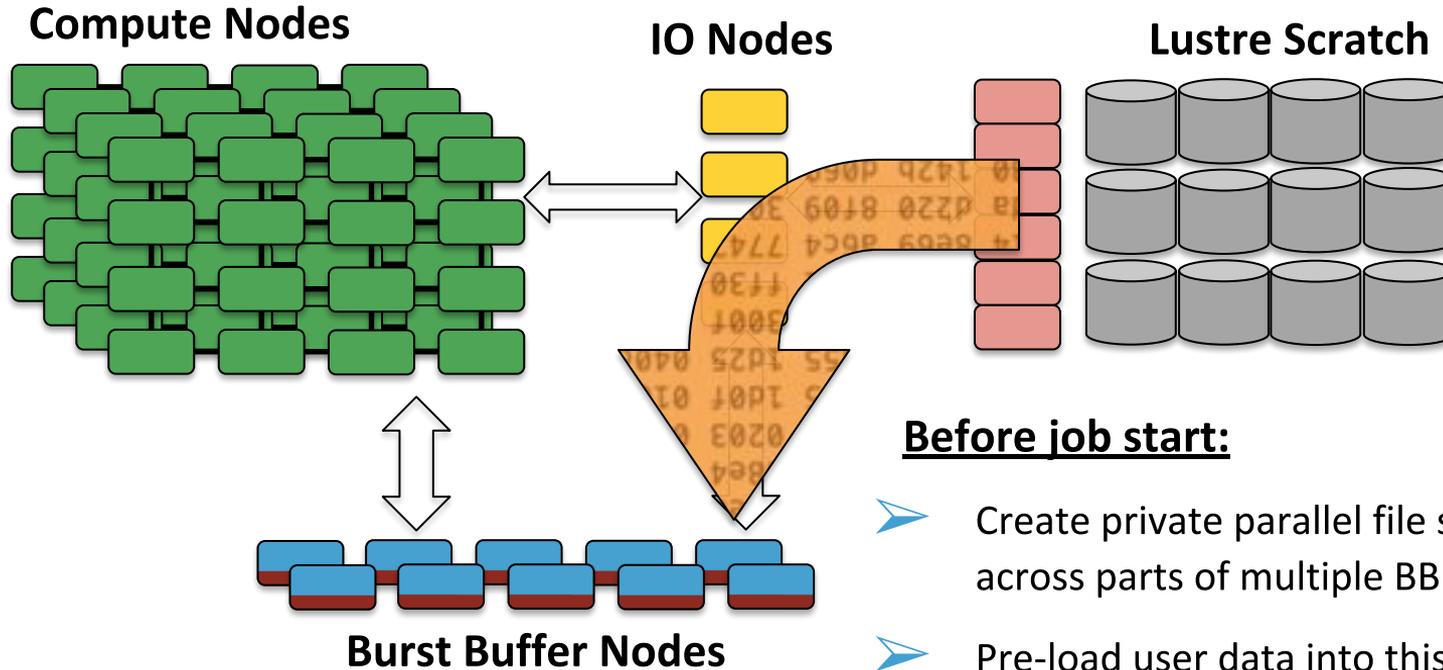
Cori's Data Paths



When submitting job, request:

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

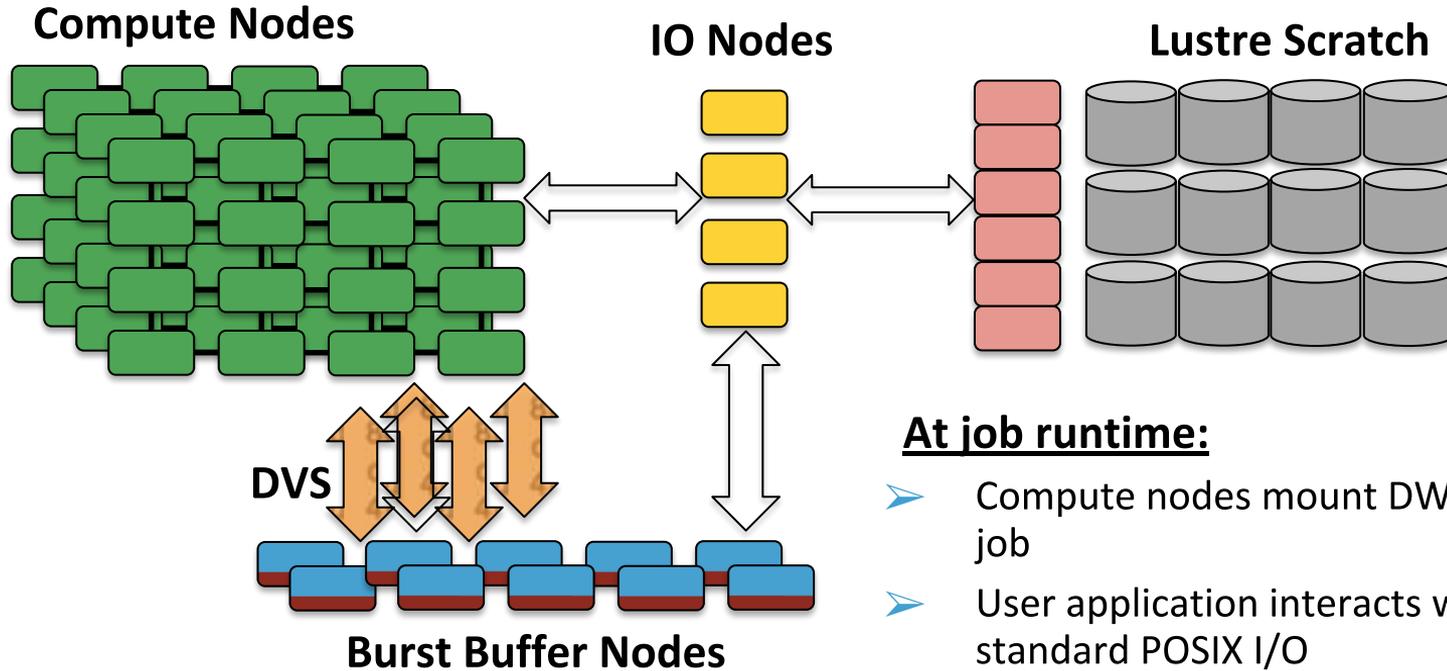
Cori's Data Paths



Before job start:

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

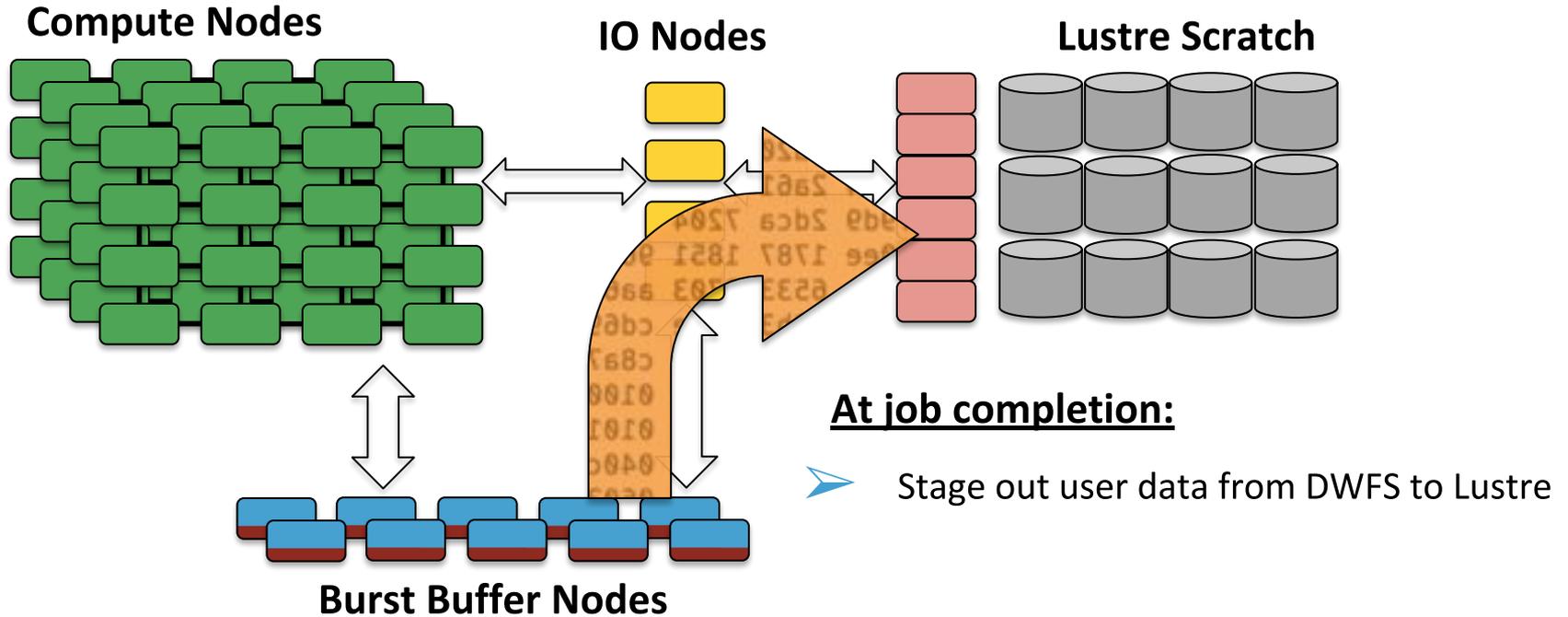
Cori's Data Paths



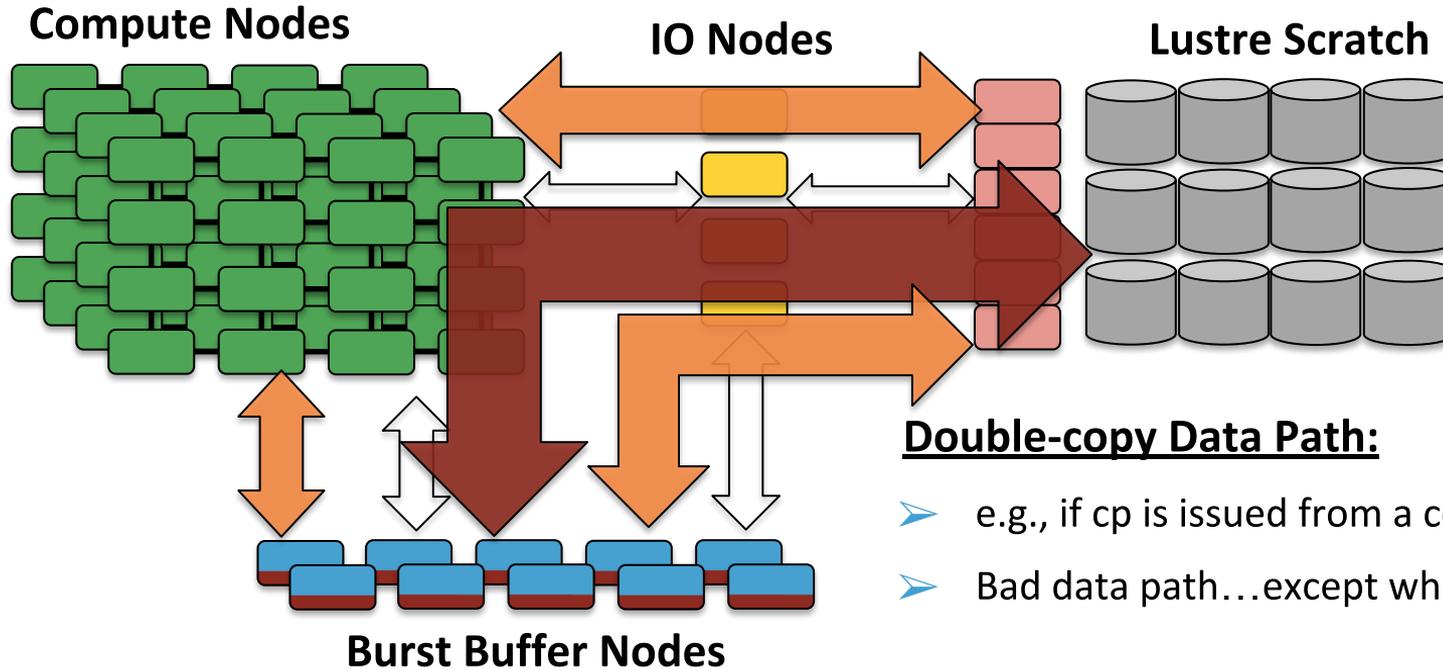
At job runtime:

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

Cori's Data Paths



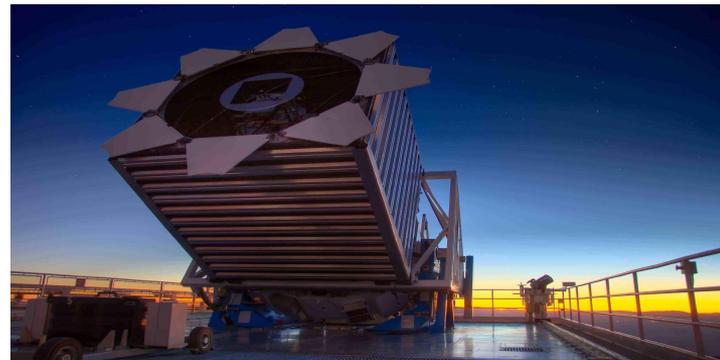
Cori's Data Paths



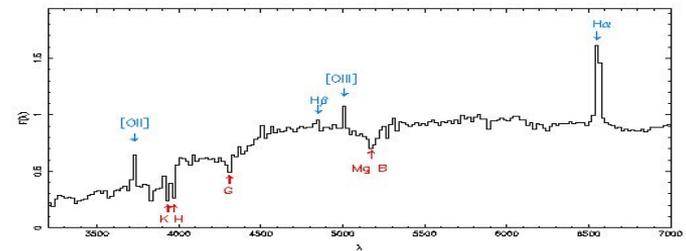
Double-copy Data Path:

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

- **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.