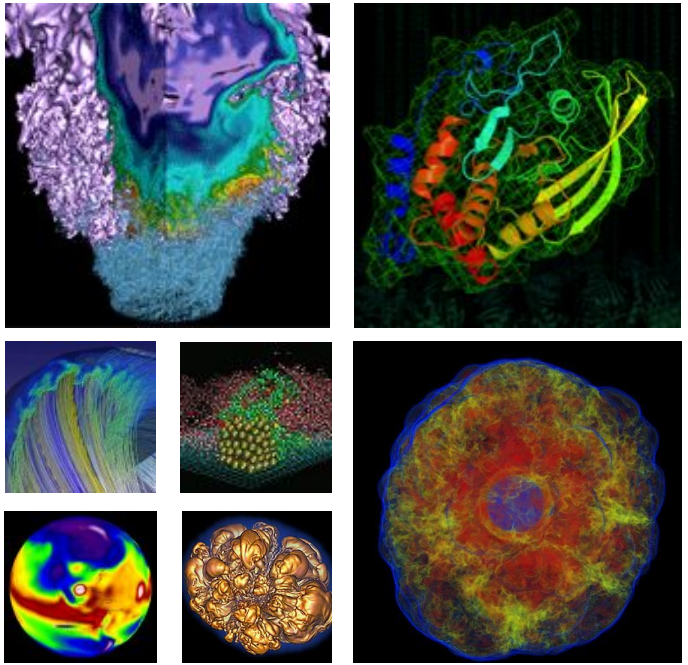
I/O and Burst Buffer



NERSC

Quincey Koziol
Data Analytics Service Group

January 25th, 2019

New User Training

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline

- **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
  - **How to use**: Stage in/out
  - **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

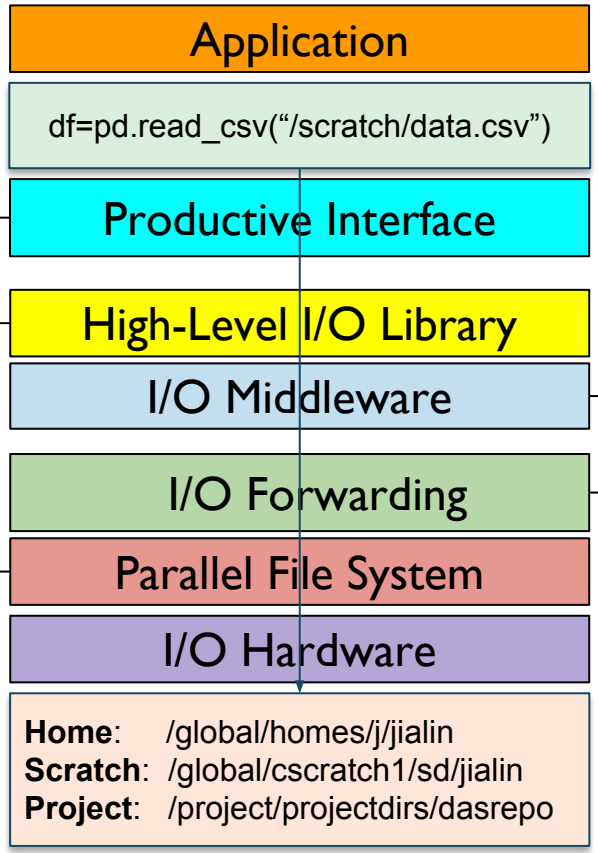
Python, Spark, TensorFlow

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

Based on Jialin Liu, Philip Carns, and Rob Ross' slides

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 IO

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

Collective IO

Coding Efforts

```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)
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
        dset[start:end,:] = temp
comm.Barrier()
timeend=MPI.Wtime()
f.close()
```

```c
#include "stdlib.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();
}
```

H5py vs. HDF5 Performance

H5Py Performance / HDF5 Performance

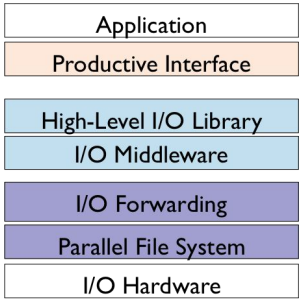
Questions: When you gain the productivity, how much performance you can afford to lose?

		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%

HDF5 vs. H5py: http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/h5py/

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

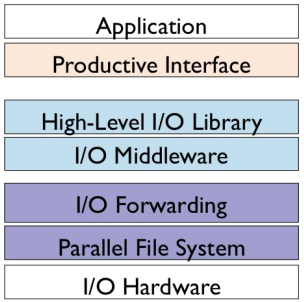
High Level I/O Libraries: HDF5

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

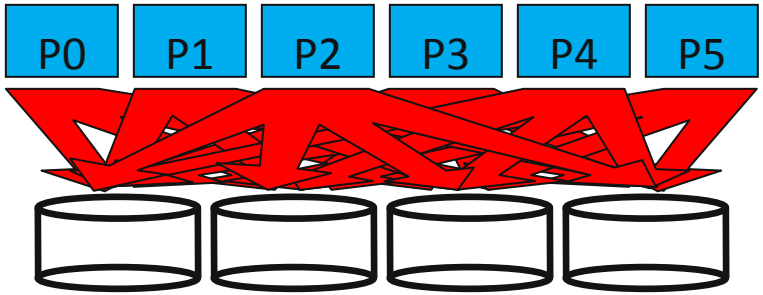
```
MPI_Init(&argc, &argv);

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();
```

I/O Middleware

- Why additional I/O Software?
  - Additional I/O software provides improved performance and usability over directly accessing the parallel file system.
  - Reduces or (ideally) 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)
  - System for encoding files in a portable format (external32)



Independent and Collective I/O

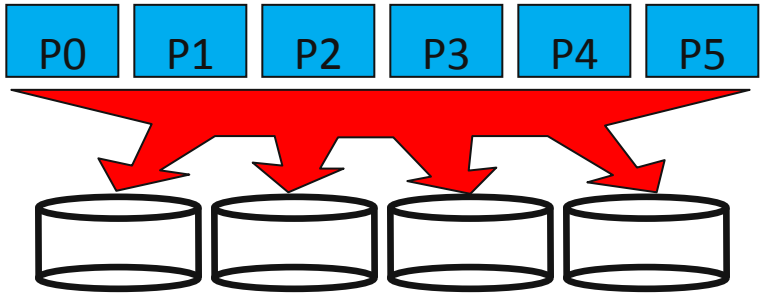


Independent I/O

- 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



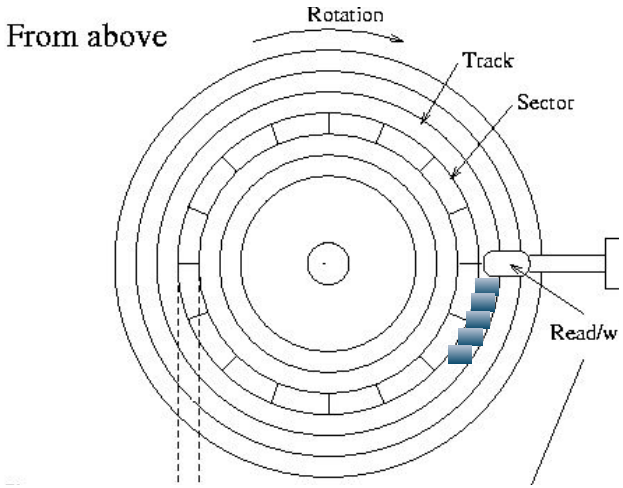
Collective I/O

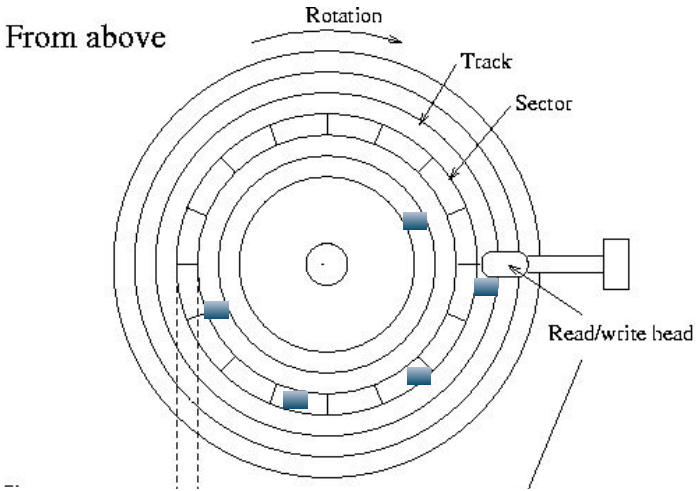
- Collective I/O is coordinated access to storage by a group of processes
  - Collective I/O functions are called by all processes participating in I/O
- Why use collective I/O
  - Allows I/O layers to know more about access as a whole, more opportunities for optimization in lower software layers, better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





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

- 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

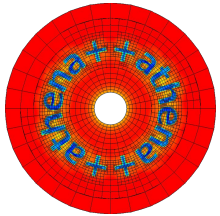
Darshan Log

- Location: /global/cscratch1/sd/darshanlogs/2019/1/25/
- Statistics: 5000~ logs per day
- Format: Username_Jobname_idSlurm_JobId_xxx.darshan
- Example: zulissi_vasp_std_id16178922_11-4-71121-6884128420676186490_3.darshan

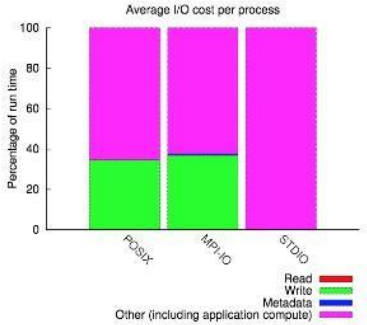
Darshan Command

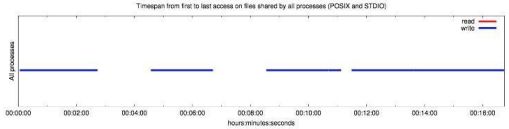
- *darshan-job-summary.pl* xxxx.darshan
- *darshan-summary-per-file.sh* xxxx.darshan

Success Story: Athena's I/O



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





IO Analysis with Darshan

"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

darshan-job-summary.pl darshan_log output.pdf

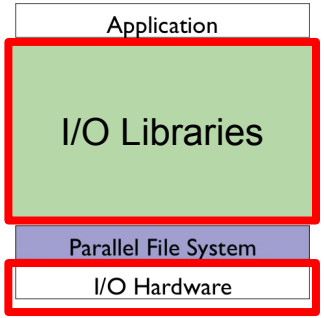
Darshan: http://www.nersc.gov/users/software/performance-and-debugging-tools/darshan/
HDF5: http://www.nersc.gov/users/data-analytics/data-management/i-o-libraries/hdf5-2/
Athena: https://princetonuniversity.github.io/athena/

What's Next

- 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

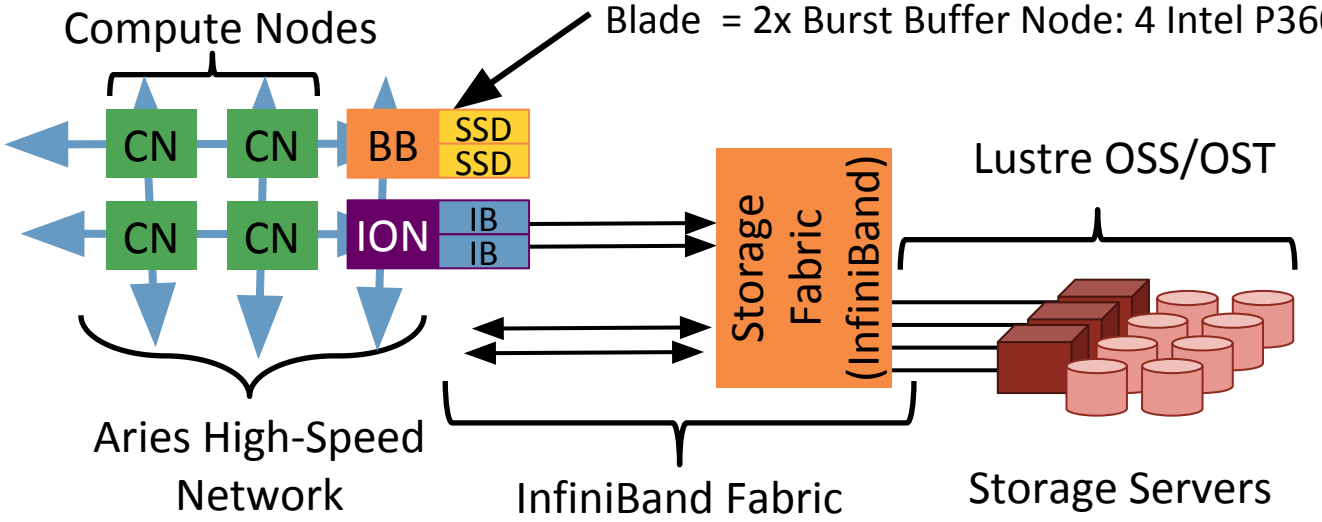
File system is also important: Lustre / GPFS / HPSS/ DataWarp

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

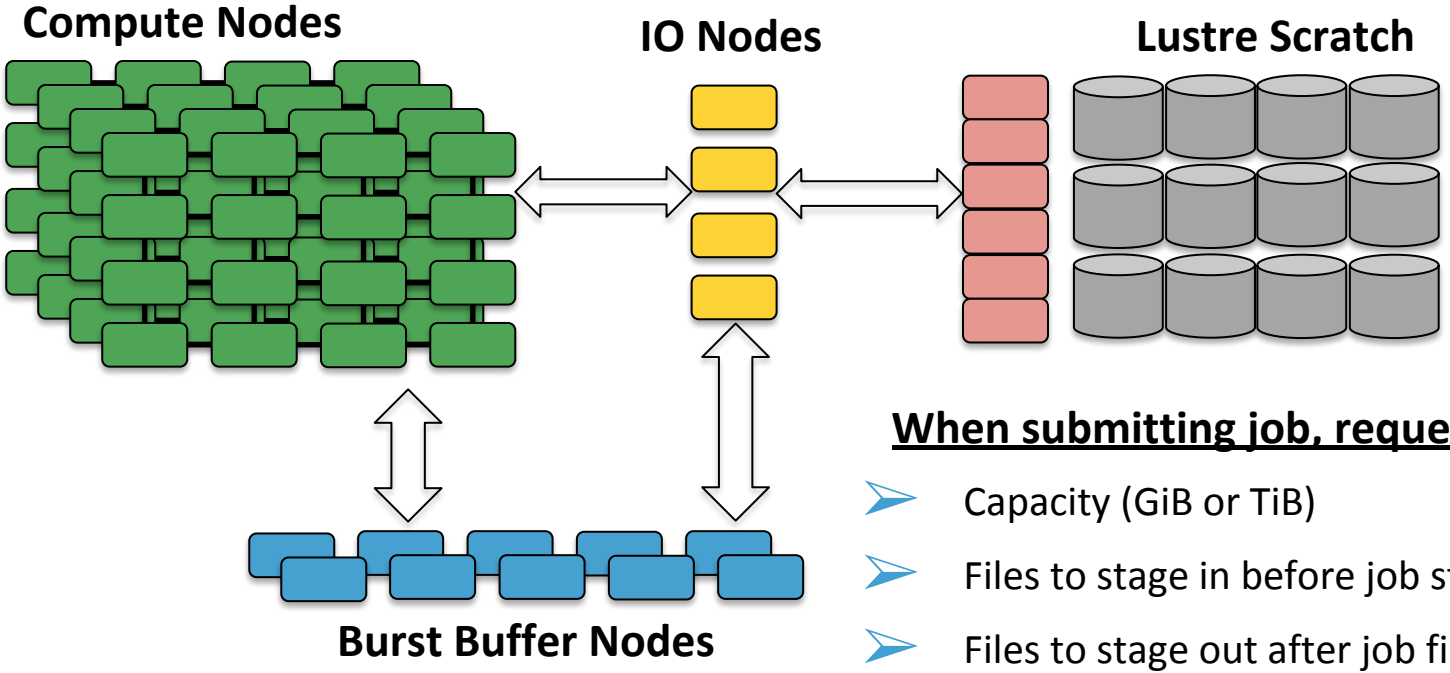


	Burst Buffer	Lustre
Nodes	288	248
Capacity (PB)	1.8	28

- 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)

Burst Buffer: http://www.nersc.gov/users/computational-systems/cori/burst-buffer/

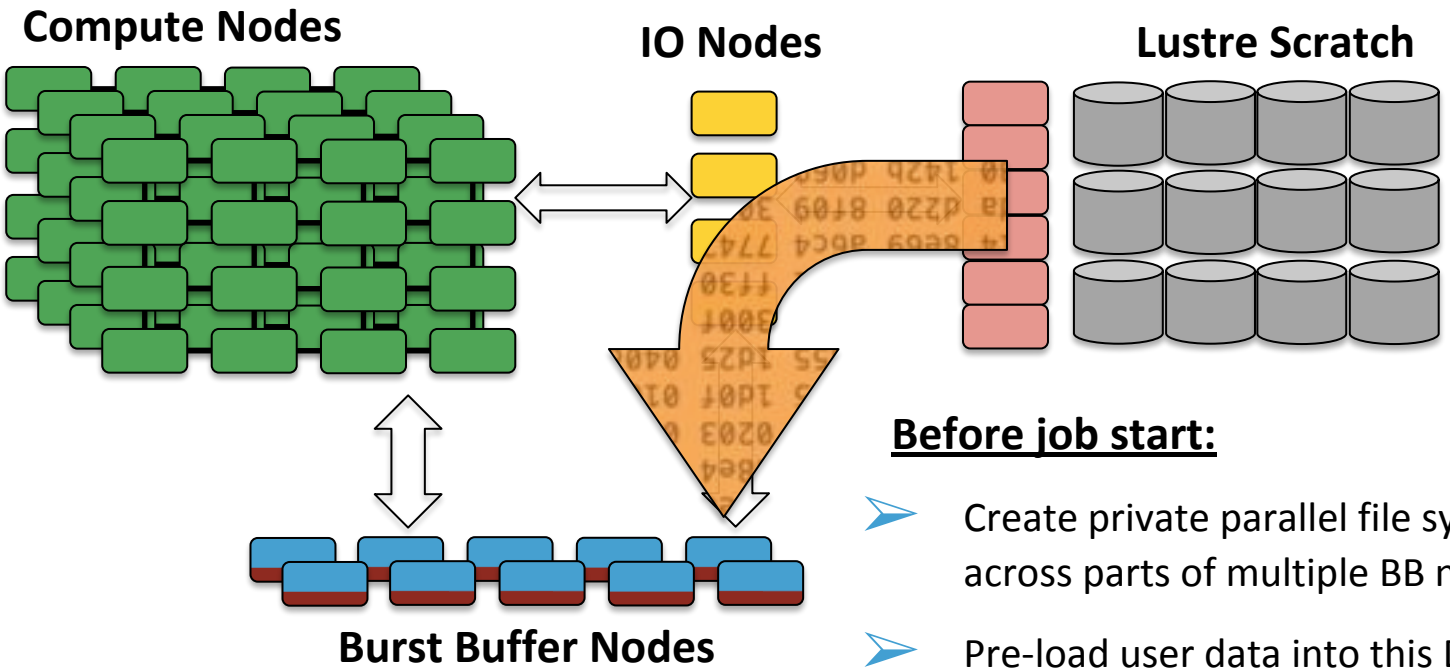
Cori's Data Paths



When submitting job, request:

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

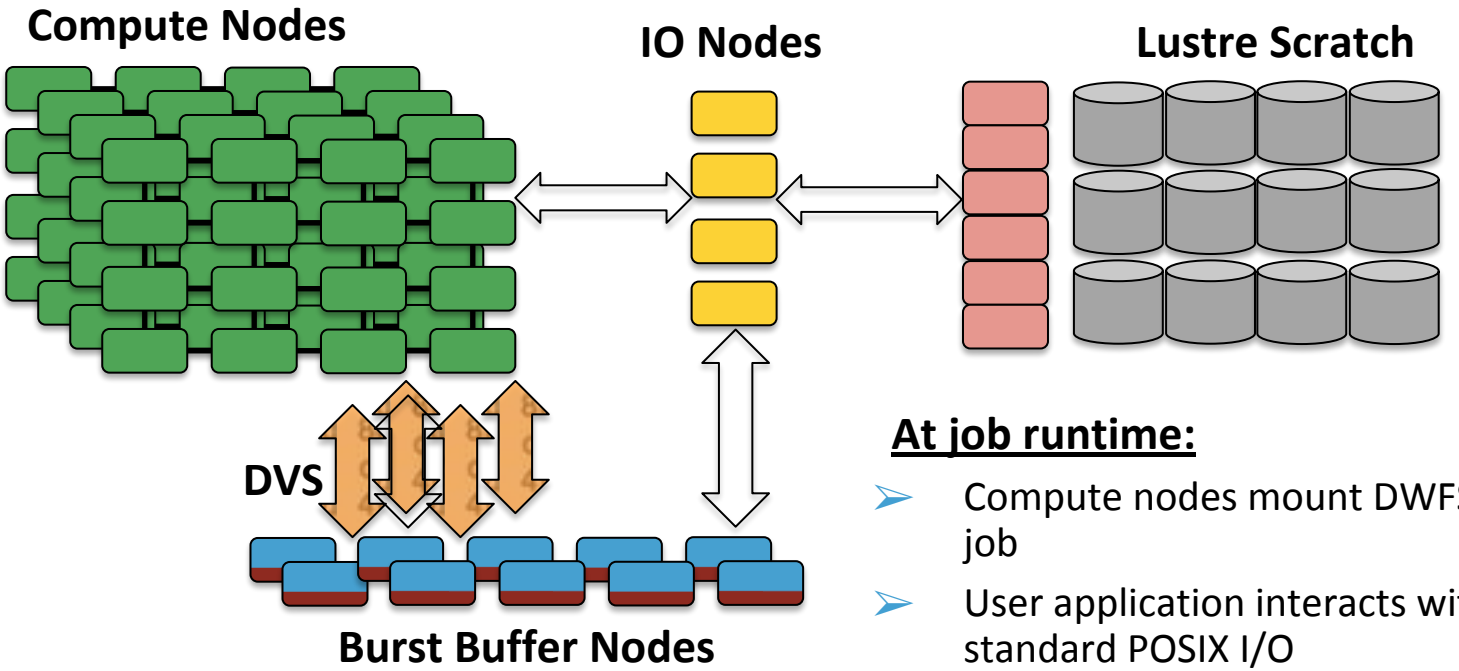
Cori's Data Paths



Before job start:

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

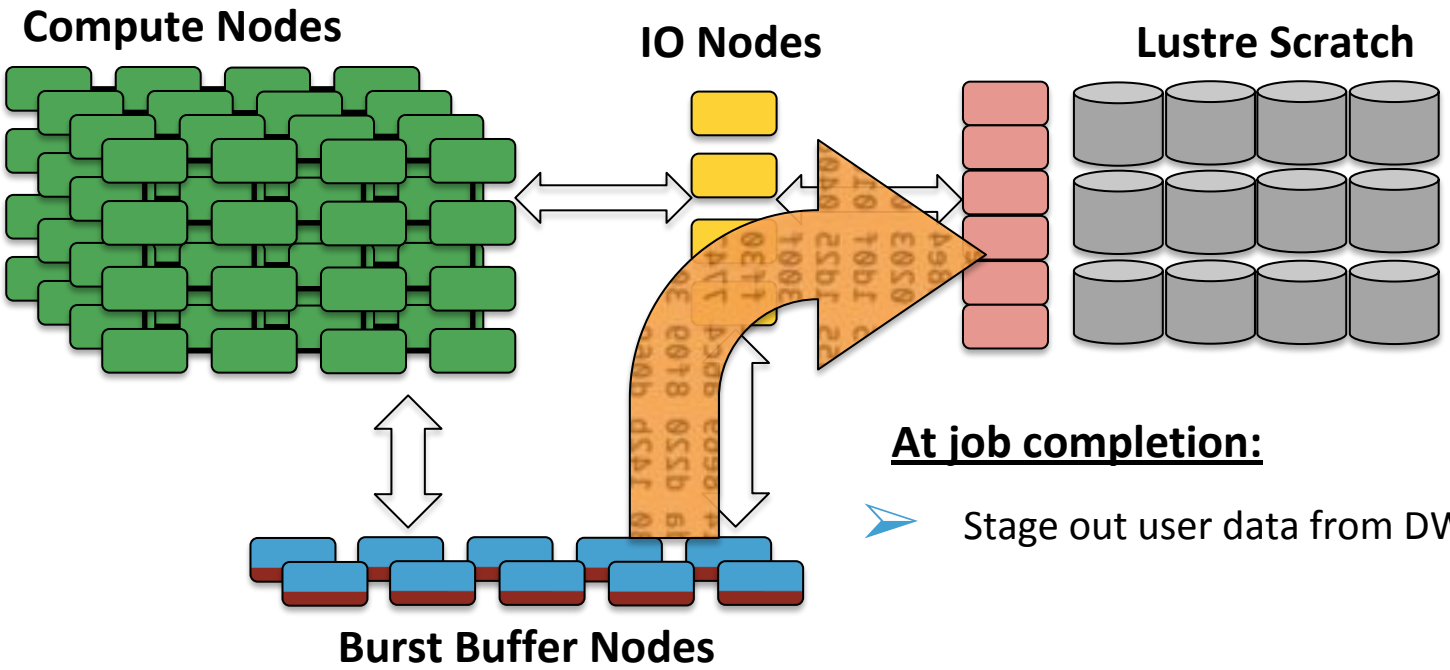
Cori's Data Paths



At job runtime:

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

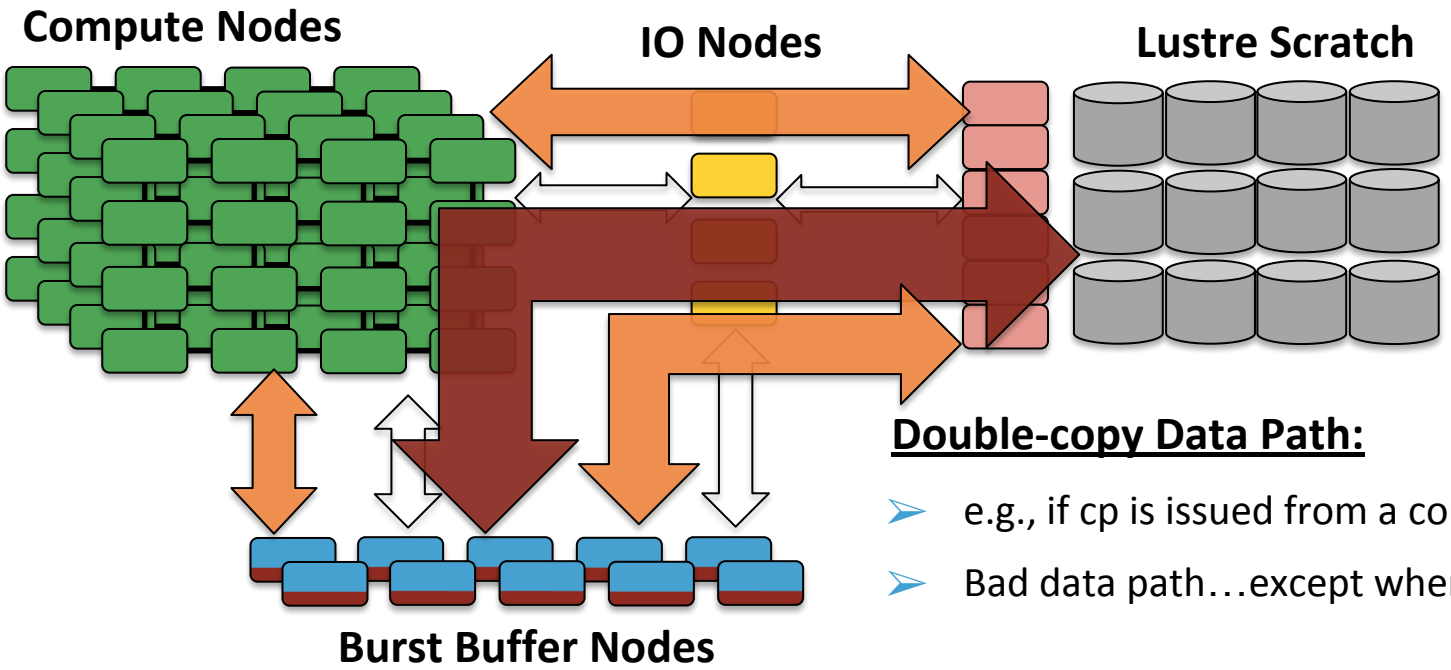
Cori's Data Paths



At job completion:

- Stage out user data from DWFS to Lustre

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

Striping, Granularity and Pools

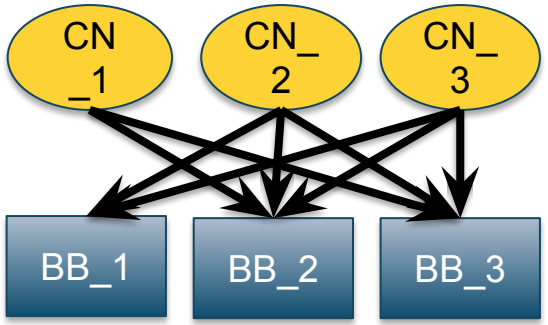
- Capacity
  - How much Burst Buffer resources, e.g., 1000GB
- Granularity
  - Minimum allocation on each Burst Buffer node
  - 20GiB by default
- Access Mode and Type
  - Mode: Striped vs. Private
  - Type: Scratch
- Put all together:
  - `#DW jobdw capacity=1000GiB access_mode=striped type=scratch`

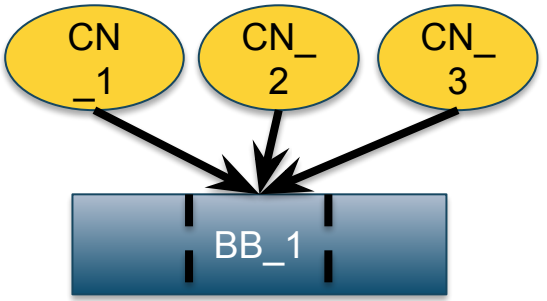
Two kinds of DataWarp Instances

- **"Instance": an allocation on the BB**
- **Can it be shared? What is its lifetime?**
  - **Per-Job Instance**
    - Can only be used by job that creates it
    - Lifetime is the same as the creating job
    - Use cases: PFS staging, application scratch, checkpoints
  - **Persistent Instance**
    - Can be used by any job (subject to UNIX file permissions)
    - Lifetime is controlled by creator
    - Use cases: Shared data, PFS staging, Coupled job workflow
    - ***NOT for long-term storage of data!***

Two DataWarp Access Modes

- Striped ("Shared")
  - Files are striped across all DataWarp nodes
  - Files are visible to all compute nodes Aggregates both capacity and BW per file
  - One DataWarp node elected as the metadata server (MDS)
- Private
  - Files are assigned to one or more DataWarp node (can chose to stripe)
  - File are visible to *only the compute node that created them*
  - Each DataWarp node is an MDS for one or more compute nodes





How to use DataWarp

- **Principal user access: SLURM Job script directives: #DW**
  - Allocate job or persistent DataWarp space
  - Stage files or directories in from PFS to DW; out DW to PFS
  - Access BB mount point via $DW_JOB_STRIPED, $DW_JOB_PRIVATE, $DW_PERSISTENT_STRIPED_name
- **User library API – libdatawarp**
  - Allows direct control of staging files asynchronously
  - C library interface
  - https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-8
  - https://github.com/NERSC/BB-unit-tests/tree/master/datawarpAPI

How to Use Burst Buffer

```
#!/bin/bash
#SBATCH -q regular -N 10 -C haswell -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \ type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \  type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- ‘type=scratch’ – duration just for compute job (i.e. not ‘persistent’)
- ‘access_mode=striped’ – visible to all compute nodes and striped across multiple BB nodes
- Data ‘stage_in’ before job start and ‘stage_out’ after

Benchmark Performance on Cori

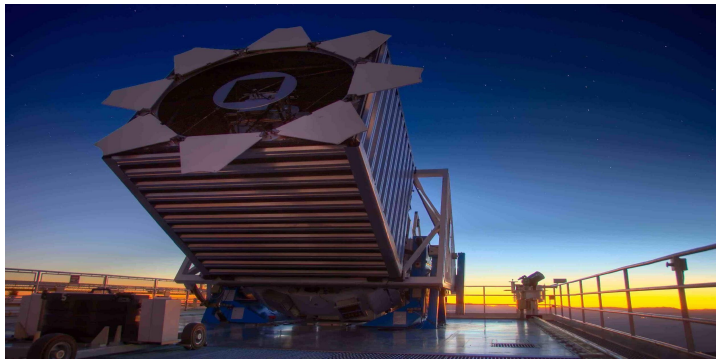
- **Burst Buffer is now doing very well against benchmark performance targets**
  - Out-performs Lustre significantly
  - (One of the) fastest I/O system in the world!

	IOR Posix FPP		IOR MPIO Shared File		IOPS	
	Read	Write	Read	Write	Read	Write
Best Measured (287 Burst Buffer Nodes : 11120 Compute Nodes; 4 ranks/node)*	1.7 TB/s	1.6 TB/s	1.3 TB/s	1.4 TB/s	28M	13M

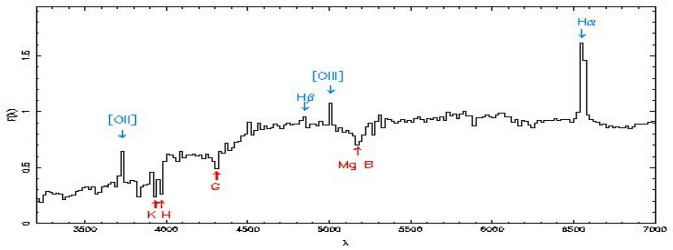
**Bandwidth tests: 8 GB block-size 1MB transfers IOPS tests: 1M blocks 4k transfer*

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.