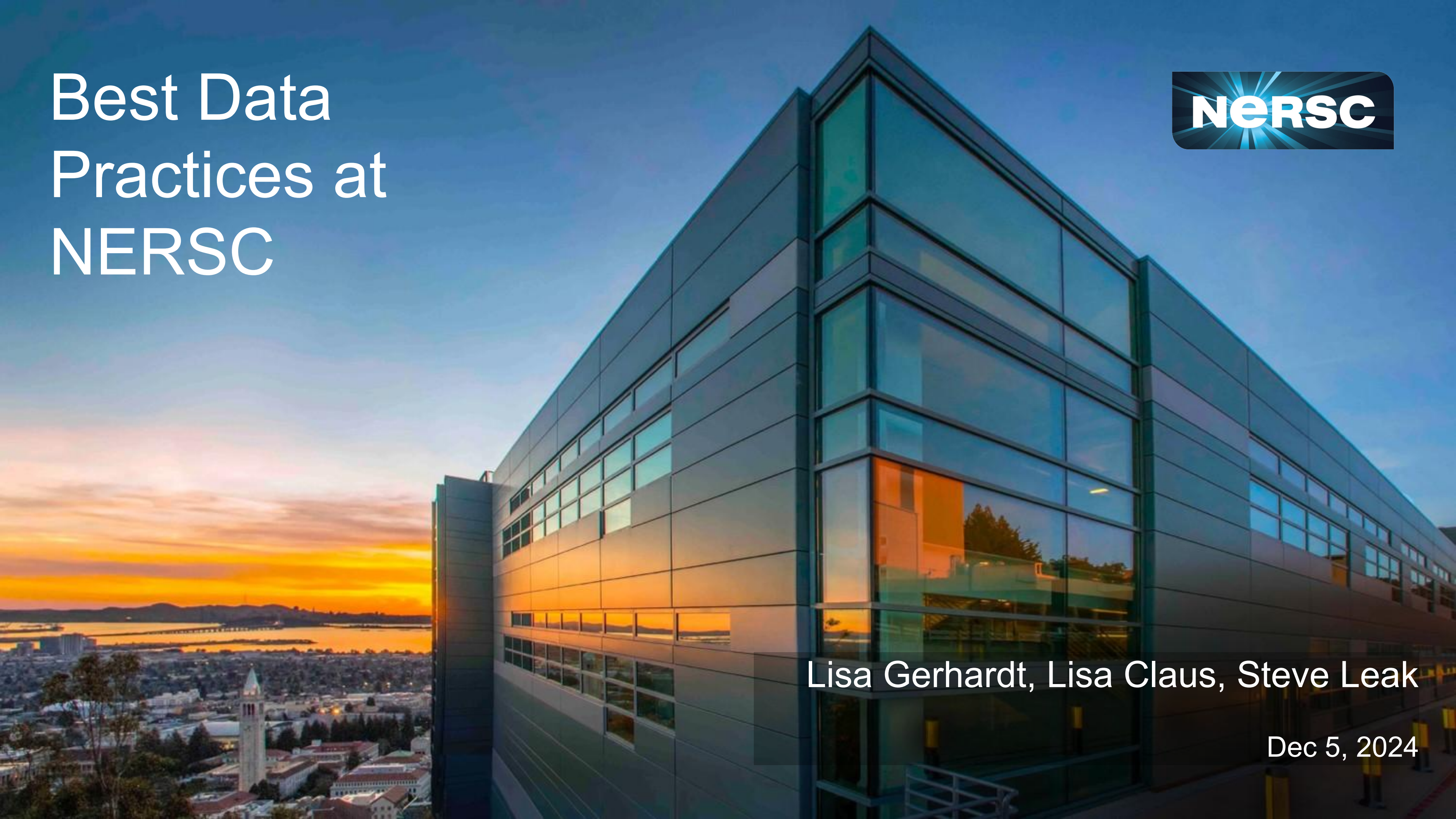
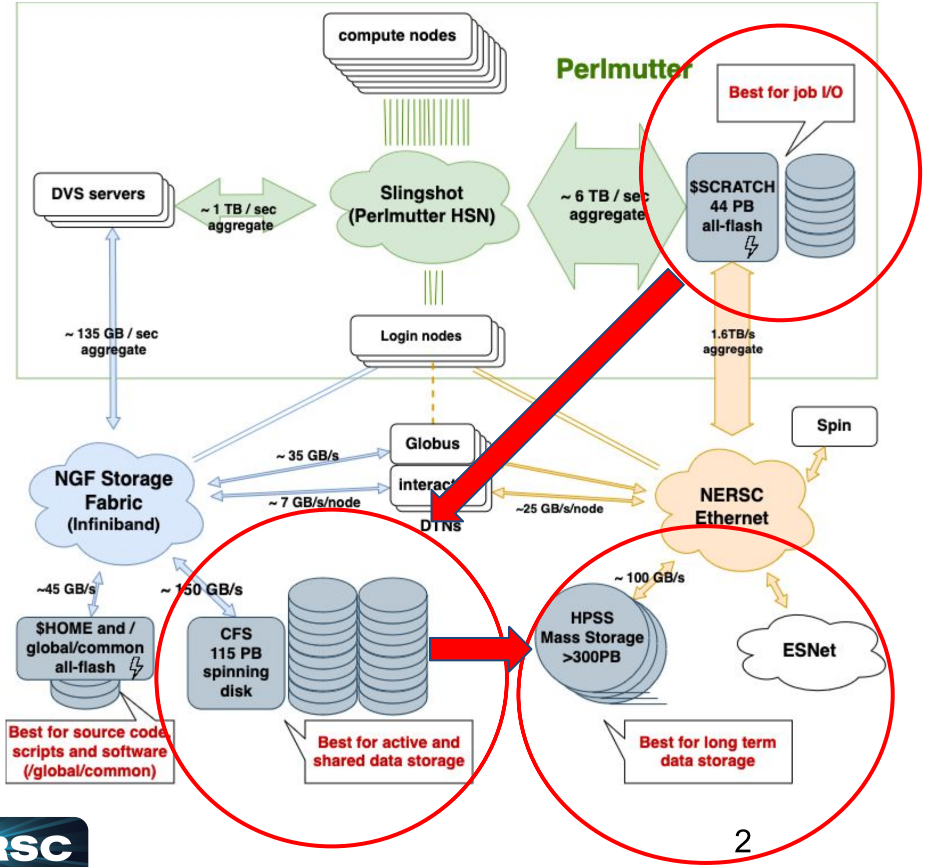


Best Data Practices at NERSC

Lisa Gerhardt, Lisa Claus, Steve Leak

Dec 5, 2024

Managing Data at NERSC



Most common workflow:

- Data created on SCRATCH by jobs
- Moved to CFS for medium-term analysis
- Moved to HPSS for long-term storage

It's **every user's** responsibility to manage their data!

Learning objectives

- NERSC file storage systems
  - Which systems are best suited to which usage scenarios
- Data management mechanisms
  - Best-practices to most-effectively manage your data
- Available tools
- Q&A

A Few Acronyms Defined

- **IO (also I/O)**: creating, reading, writing, listing, moving, copying and deleting files and directories.
- CFS (Community File System): large-scale file system for sharing project data
- HPSS (High Performance Storage System): huge tape archive at NERSC
- **DTN** - Data Transfer Node - optimized for data movement
- **DVS** - Data Virtualization Service (DVS) is an I/O forwarding service that works by projecting a parallel file system, eg CFS, /global/common and $HOME, to compute nodes
- UNIX - Unix is a family of multitasking, multi-user computer operating systems. Linux is a form of Unix.
- POSIX - Portable Operating System Interface, is a set of standards that define how operating systems interact with applications (Linux complies with POSIX)
- ACLs - Access Control Lists - fine-grained control over who can read/write a file, via setfacl/getfacl
- OST - Object Storage Targets (disks)

Overview NERSC storage systems

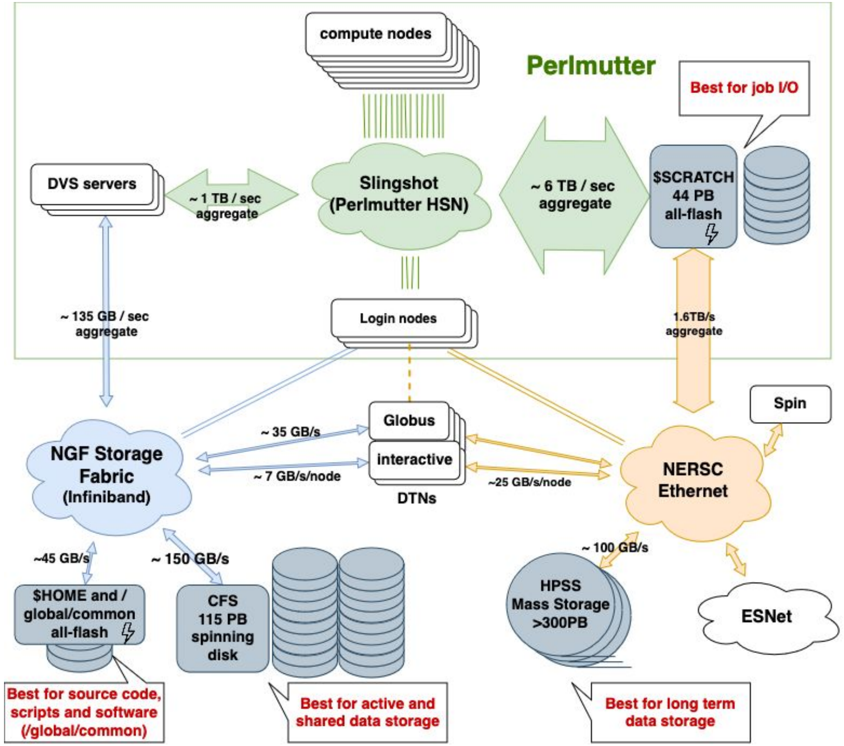
- $SCRATCH - Perlmutter Scratch
  - Every User has its personal space on SCRATCH: /pscratch/sd/FirstLetterOfUserName/YourUserName
  - Perlmutter scratch is available to all Perlmutter compute nodes and is **tuned for high performance**
- CFS - Community File System
  - Every NERSC project has an associated Community directory and Unix group: $CFS/<your_project_name>
  - CFS allows **sharing of data** between users, systems, and the "outside world"
- HPSS - High Performance Storage System
  - By default every user has an HPSS account
  - HPSS is intended for **long term storage** of data that is not frequently accessed
- HOME
  - Every User has its **personal small space** on $HOME
- Global Common File System
  - Every NERSC project has a directory on /global/common/software/<your_project_name>
  - Global Common is **optimized for software installation**

Data Lifecycle at NERSC

- Most common pattern is data starts on $SCRATCH and moves to CFS and then to HPSS
- $SCRATCH is for data being actively computed against
- CFS is for data that will be accessed in this year or maybe the next
- HPSS is for important data you need to keep long term
- As performance increases, capacity decreases and data must migrate to another layer

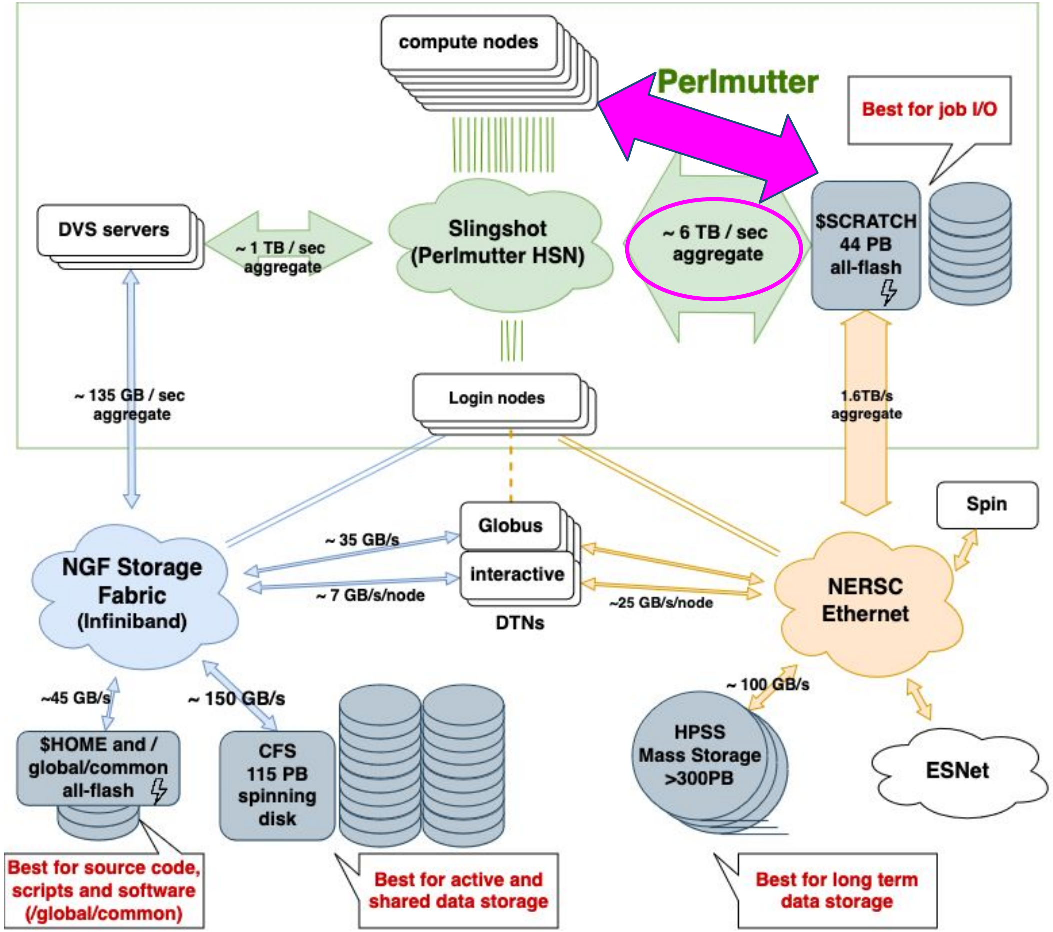
Choosing the right storage for your data

- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!



Job I/O to $SCRATCH

- Short, quick path between computes and a big, fast, filesystem
- Supports parallel I/O (file locking)
- Short-term storage!



$SCRATCH

- Big: 20TB soft quota, 30TB hard quota
  - Over soft quota: job won't start
  - Over hard quota: writes fail
- Fast: Highly parallel, all-flash, 6TB/s aggregate bandwidth
- Full POSIX:
  - File locking (for parallel I/O)
  - MPI-IO
  - ACLs
- Handles big and small files and I/O operations well
  - input and output data
  - config files and scripts
  - compilation

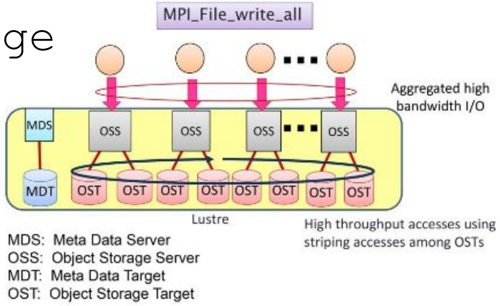
- Not huge: full scientific datasets can be hundreds or thousands of TB - **$SCRATCH is for I/O, not storage**
- No backups:
  - Anything deleted (or purged) is **gone**
  - In event of catastrophic disk crash, data may not be recoverable
- **Purged**

$SCRATCH tips

- Optimize performance with striping
  - https://docs.nersc.gov/performance/io/lustre/#nersc-file-striping-recommendations
  - Splits the file across multiple OSTs (disks)
  - By default, data on 1 OST, ideal for small files and file-per-process I/O
  - Single shared-file I/O should be striped according to its size
  - Helper scripts

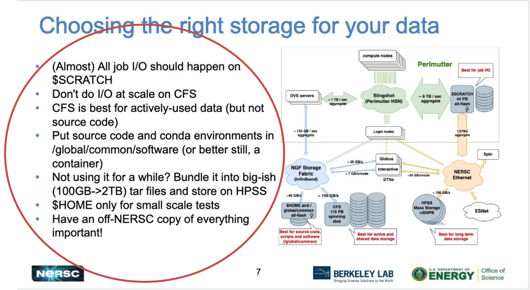
    `stripe_small`, `stripe_medium`, `stripe_large`
  - Manually query with

    `lfs getstripe <path>`
  - Set striping on a directory
    - New files will automatically pick it up
    - Copy files in to inherit the striping



1. MPI-IO on Lustre: https://www.sys.r-ccs.riken.jp/ResearchTopics/fio/mpiio

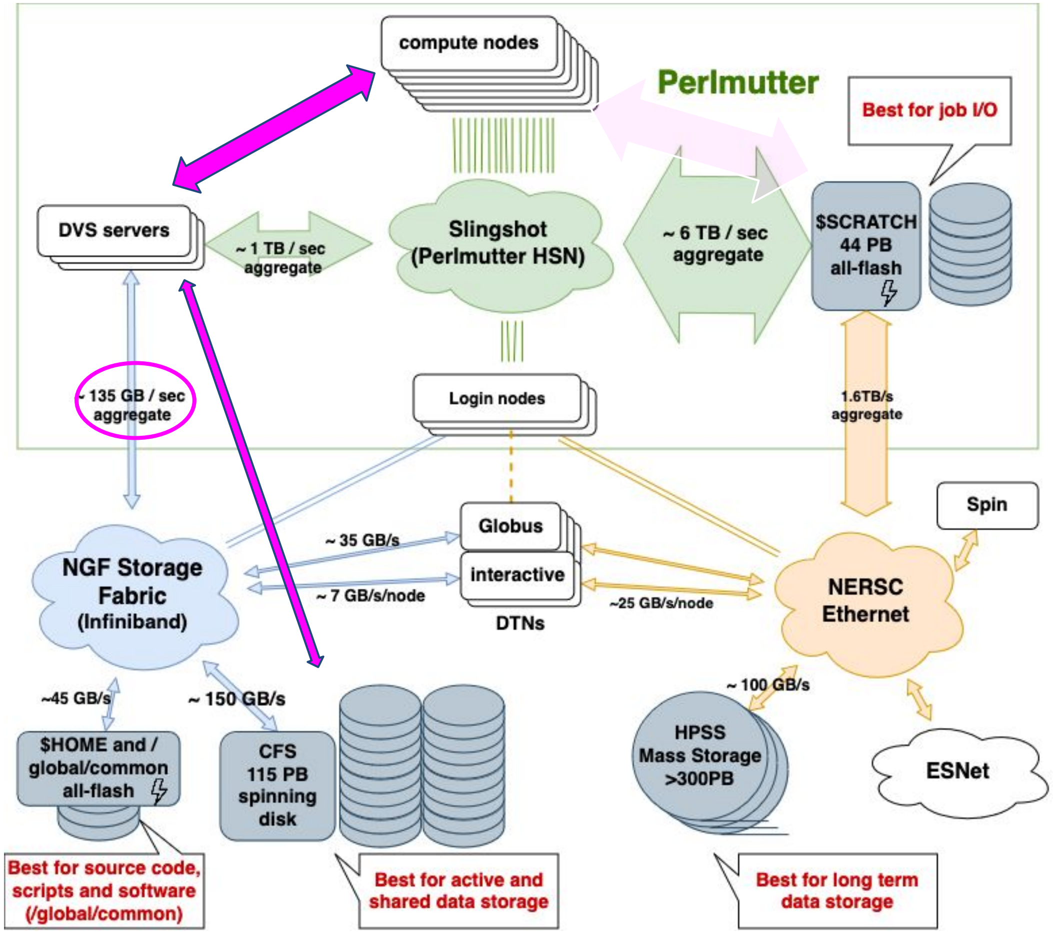
TL; DR



- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!

Data on CFS

- Capacity-oriented filesystem, huge, robust
- Longer, indirect (**via DVS**) path to compute nodes
  - CFS is not suited for job I/O at scale



CFS

- Huge: Currently 114 PB, 33 PB more coming soon
  - Large block size: great for files >>1MB
- Robust:
  - multiple layers of redundancy for reliability
  - daily snapshots retained for 7 days - if the file existed yesterday, you can recover from an accidental deletion
- Never purged, readily accessible
- Projects can split their space allocations between multiple directories and give **separate** working groups **separate** quotas
- Data dashboard: https://docs.nersc.gov/filesystems/quotas/#the-data-dashboard
- Full POSIX when directly mounted
  - ie login nodes, DTNs (but not Perlmutter compute nodes)

- Configured for capacity over performance
  - (Still *pretty* fast, but not $SCRATCH fast)
  - Large block size - inefficient for small files, eg source code
- Not directly mounted on Perlmutter compute nodes
  - Mounted via an I/O forwarding service named DVS (more on that next), which imposes some constraints - not suitable for most job I/O
- Not backed up - make sure you have a copy of data, somewhere else

Making Sharing Data Easier on CFS

- Sharing in space with large groups is hard, we made tools

Data Dashboard Demo

Data Dashboard

Showing disk space and inode usage for global directories at NERSC to which you have access as PI, PI proxy, or user (includes /cfs, /dna, and /projectb)

atlas directory in /cfs

Toggle Usage Details | My Files and Dirs | Browse

840.33 of 919.00 TB allocated (91%) — 0 … 100%

64.57 of 69.00 M inodes allocated (94%) — 0 … 100%

bbtools directory in /cfs

Toggle Usage Details | My Files and Dirs | Browse

19.07 of 20.00 TB allocated (95%) — 0 … 100%

0.78 of 1.00 M inodes allocated (78%) — 0 … 100%

CAL directory in /cfs

Toggle Usage Details | My Files and Dirs | Browse

6.50 of 8.00 TB allocated (81%) — 0 … 100%

1.22 of 8.00 M inodes allocated (15%) — 0 … 100%

carver directory in /cfs

Toggle Usage Details | My Files and Dirs | Browse

0.00 of 1.00 TB allocated (0%) — 0 … 100%

0.06 of 1.00 M inodes allocated (6%) — 0 … 100%

Adjusting CFS Quotas in IRIS (Demo)

- Projects can spread their CFS quota across multiple directories. Each row is a “top level” directory, i.e. path is “/global/cfs/cdirs/<directory>”

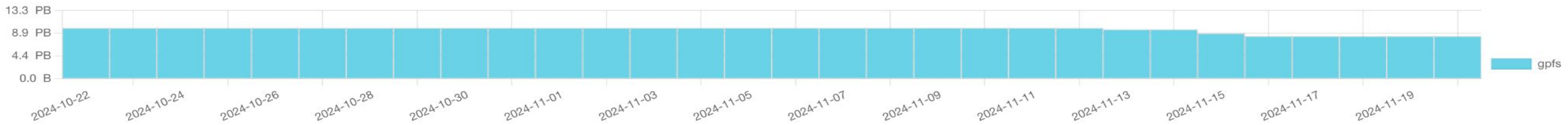
nstaff | CPU | GPU | Jobs | Storage | Roles | Groups | Details | History

CFS Storage

Storage Allocation:	14.6 PB
Storage Allocated to Directories:	11.0 PB
Unallocated Storage:	3.6 PB
Storage Used:	8.3 PB
% Storage Allocation Used:	56.6%

CFS Files

Files Allocation:	700.0 M
Files Allocated to Directories:	330.4 M
Unallocated Files:	369.6 M
Files Used:	127.3 M
% File Allocation Used:	18.2%



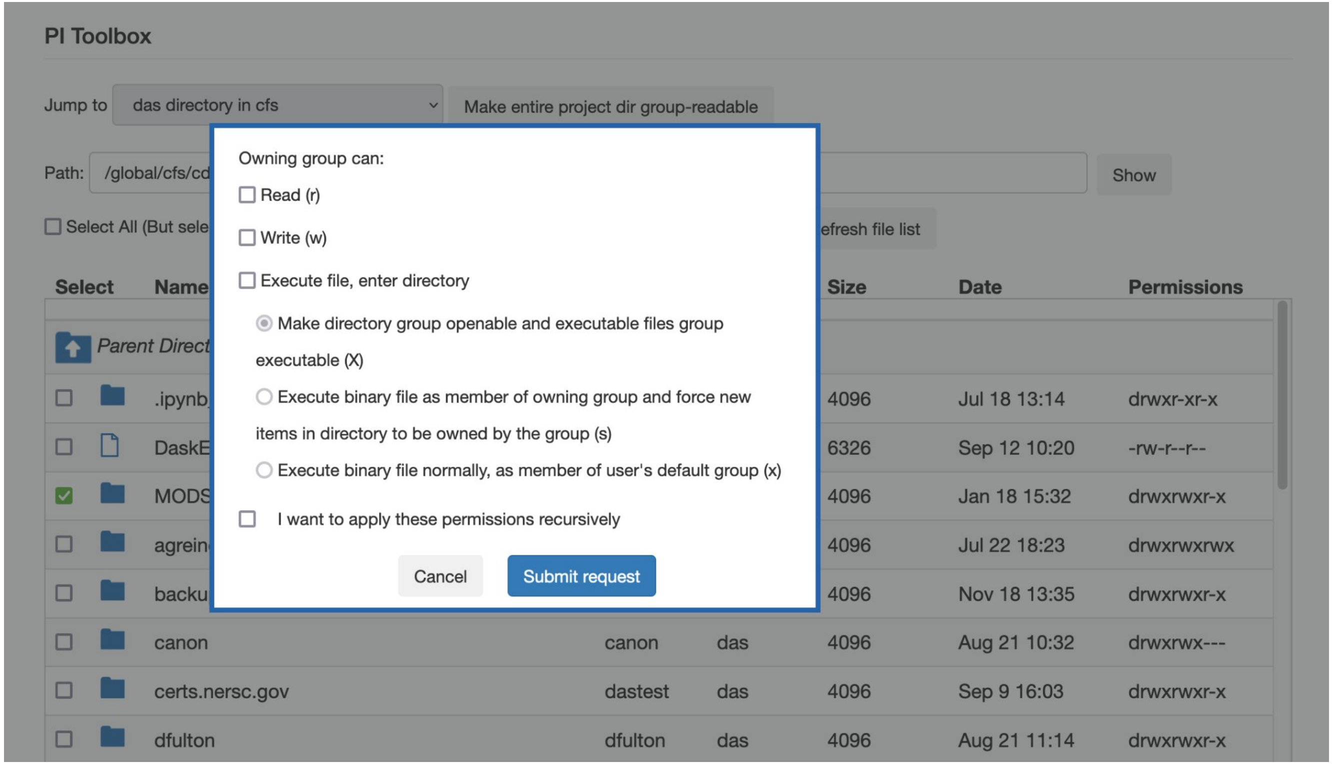
CFS Directory Usage

Directory	File System	Owner	Group	Active	Storage Used	Directory Storage Quota (TB)	% Storage Used	Files Used	Directory File Quota	% Files Used	Actions
n7impl	gpfs		n7impl		2.0 TB	2.00	101.2%	145 K	200,000	72.7%	Edit Activate
LDMS_Analy...	gpfs		ldms_ana		27.2 TB	30.00	90.5%	30 K	35,000	85.2%	Edit Archive
genomes	gpfs		genome2		3.4 TB	4.00	84.8%	216 K	400,000	53.9%	Edit Archive
ssgdata	gpfs		ssgdata		20.9 TB	25.00	83.4%	304 K	10,000,000	3.0%	Edit Archive
ldms	gpfs		nstaff		8.0 PB	10,450.00	77.9%	156 K	10,000,000	1.6%	Edit Archive
nerscweb	gpfs		nerscweb		782.3 GB	1.00	76.4%	9 M	20,000,000	44.4%	Edit Archive
nesap	gpfs		nesap		776.8 GB	1.00	75.9%	222 K	10,000,000	2.2%	Edit Archive

Update Quotas

+ New

PI Toolbox: my.nersc.gov/pitools/



A bit about DVS

- DVS is an I/O forwarder developed by Cray
  - DVS nodes mount the filesystem, and "project" it to compute nodes
- DVS is mechanism for how you access these file systems on compute nodes. It is used to access home, common, and CFS
- Designed to deliver file system contents at scale
- Long history of deployment at NERSC, went live on Perlmutter on June 8, 2023
- Used only for compute nodes, logins have a native client mount

DVS

- Can provide filesystem access to thousands of nodes
- Decouples the filesystem from issues on Perlmutter
  - Using DVS on Perlmutter has greatly improved system and file system stability
- I/O at scale to a Read/Write -mounted filesystem is problematic
  - Using a read-only mount point can alleviate this
- Does not fully support POSIX
  - No file locking (shared-file writing via MPI-IO is not safe, HDF5 will complain and fail)
  - ACLs disable caching
    - `chmod` is fine
    - `setfacl` will cause subsequent accesses to be very slow
  - No mmap

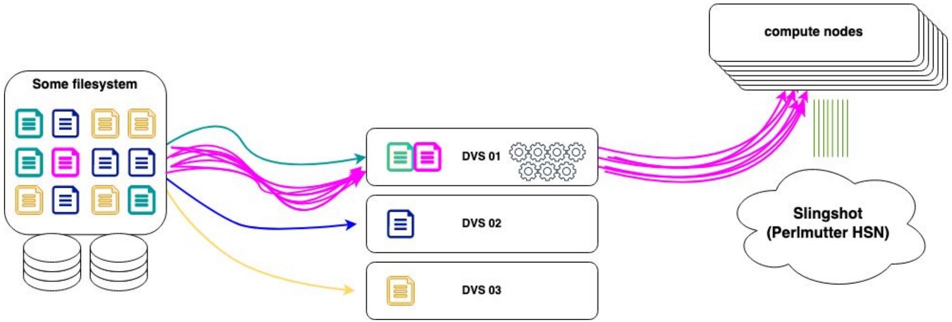
How DVS works

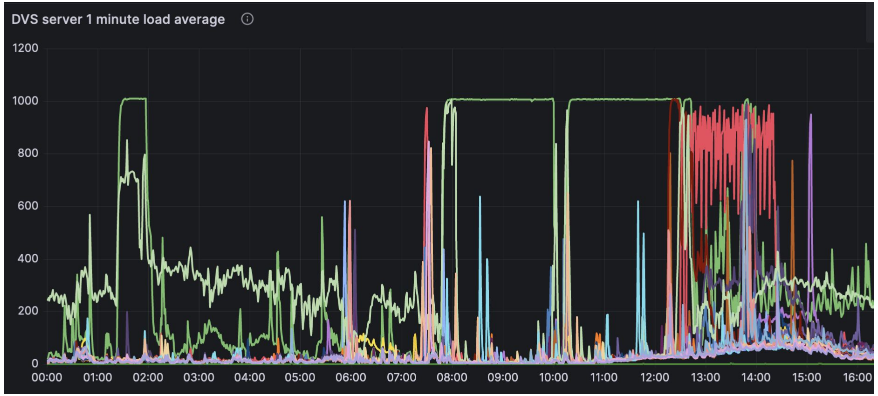
- Perlmutter has 24 gateway nodes that serve as DVS servers
- Each server can work 1000 I/O threads at once
- Can cache data to dramatically improve performance at large scales
- Two service modes:
  - Read / Write (RW): gateway server is determined when file is created (hash of inode), stays constant, zero cache
  - Read Only (RO): file can be served by all gateways, stays in cache for 30 seconds
- How to get the benefits (and avoid the limitations) of DVS:

https://docs.nersc.gov/performance/io/dvs/#best-practices-for-dvs-performance-at-scale

DVS with read-write mount ($HOME, CFS)

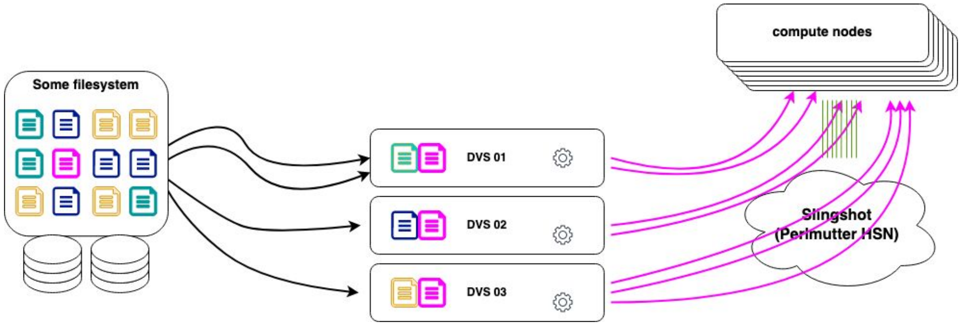
- Eg: a 100-node job using conda environment in $HOME
  - 12,800 processes all try to read `/global/homes/e/elvis/.conda`
  - No cache, so it is fetched from the filesystem 12,800 times
  - The DVS server that "owns" that file drowns under the load, while your processes wait in line
  - The job progresses only very slowly, and may fail (and other jobs using that server might be impacted too)

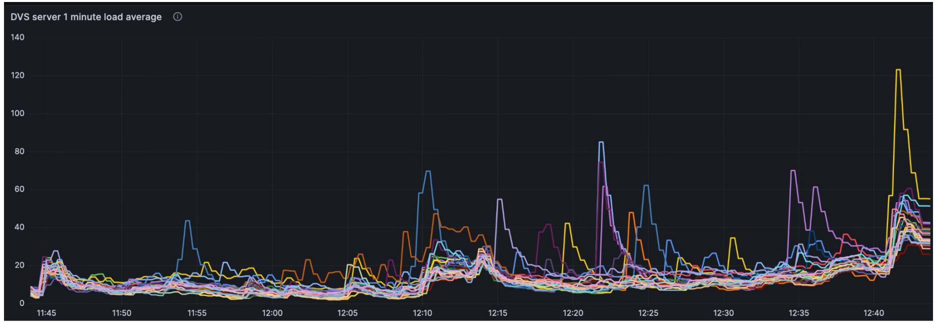




DVS with read-only mount (/global/common)

- Eg: a 100-node job using conda environment in /global/common
  - 12,800 processes are spread across 24 DVS servers
  - `/global/homes/e/elvis/.conda` gets fetched once and cached
  - Load on the DVS servers stays low
  - Load on the filesystem stays low
  - The job continues almost immediately

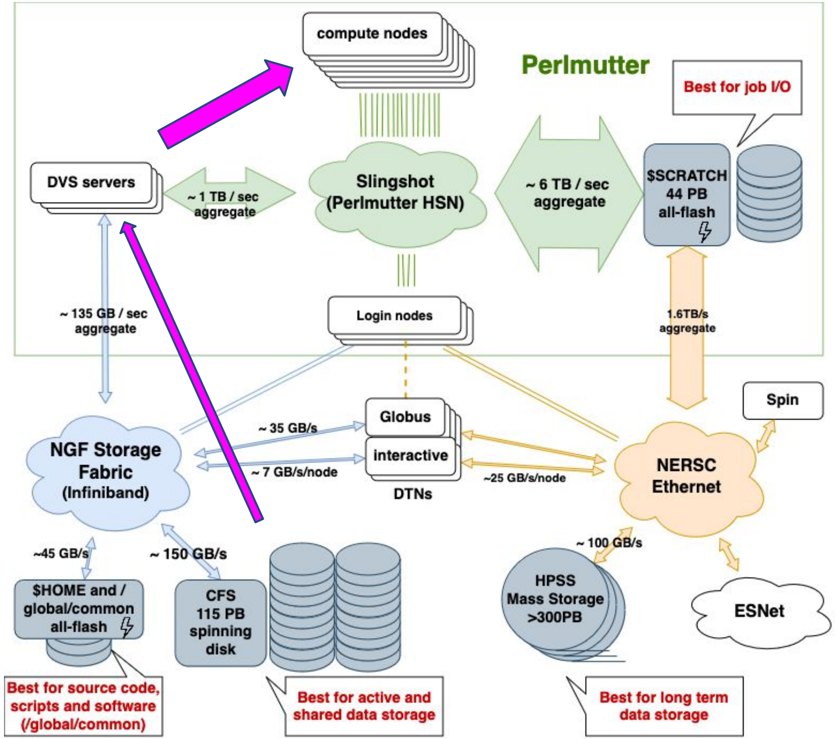




note that this y-axis goes 1/10 as high!

Read-only mount of CFS

- CFS also has a read-only mount point on Perlmutter: `/dvs_ro/cfs/cdirs/` (the RW one is `/global/cfs/cdirs`)
- $SCRATCH is still faster .. BUT if your input data is:
  - too big for $SCRATCH, and/or:
  - used by multiple people in your project
- .. then you might benefit from reading it directly from `/dvs_ro/cfs/cdirs`

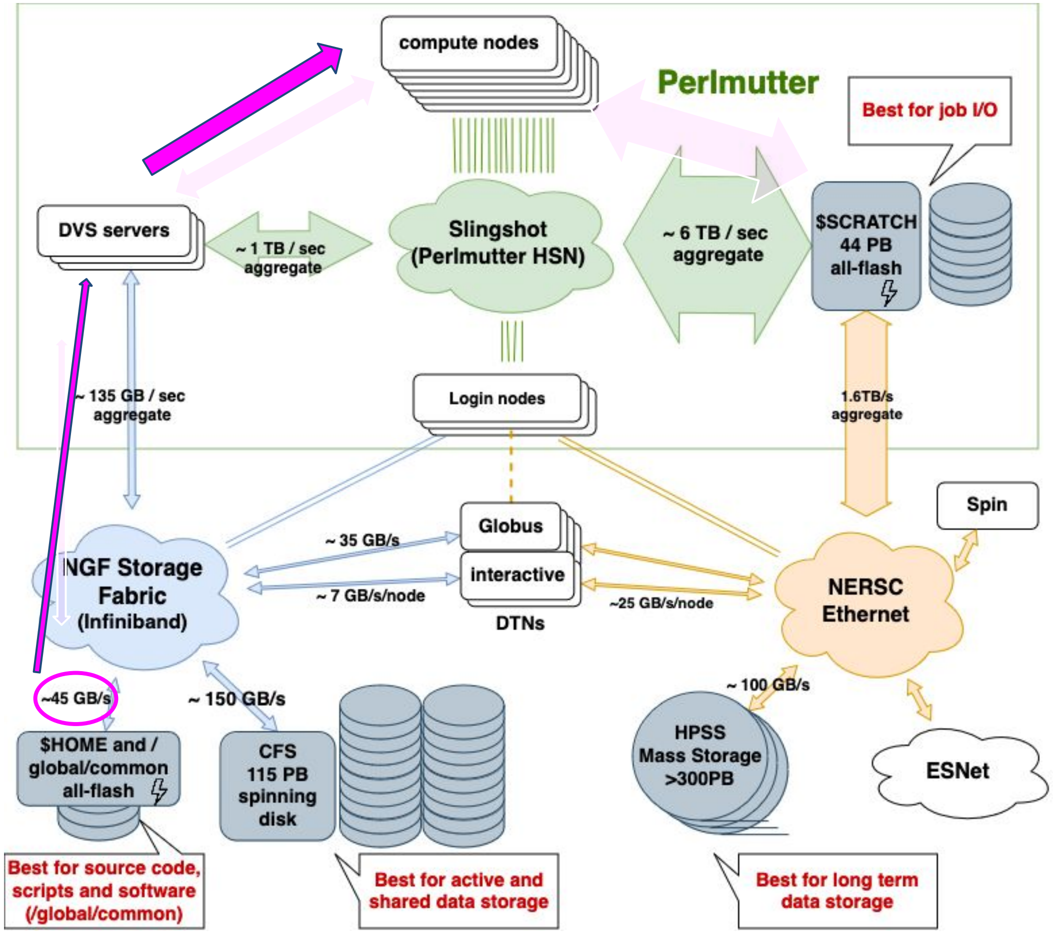


TL; DR

- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!

Software on /global/common

- Small block size, all-flash, **mounted read-only on compute nodes** (read-write on logins)
- Benefits from **DVS caching**, multiple DVS nodes



/global/common/software

- Especially good for python / conda environments!

  ```
  conda create --prefix /global/common/software/myproject/myenv
  ```

  https://docs.nersc.gov/development/languages/python/nersc-python/#moving-your-conda-setup-to-globalcommonsoftware

- Python startup involves loading lots of modules, which involves looking in all of the directories in LD_LIBRARY_PATH - *lots* of disk access
- The read-only DVS mount of /global/common/software mitigates most of this
- Related tip:
  - Don't load a conda environment at login! (via .bashrc/.bash_profile). It will be loaded for every Slurm job too.

Software in containers

- NERSC supports Shifter and Podman (newer, solves some limitations of Shifter). Both provide similar functionality to Docker
  - https://docs.nersc.gov/development/podman-hpc/overview/
  - https://docs.nersc.gov/development/shifter/how-to-use/
- How do they help?
  - Software is *in the container* - vastly reduces load on filesystem
  - Also: consistent environment each run, even if Perlmutter software stack changes -> reproducibility benefits

Podman

Shifter

Number of nodes	`$HOME`	`/global/common/software`	Shifter
1	0m4.256s	0m3.894s	0m3.998s
10	0m10.025s	0m4.891s	0m4.274s
100	0m30.790s	0m17.392s	0m7.098s
500	4m7.673s	0m48.916s	0m14.193s

Python benchmark compared by filesystem or container, over increasing node count

TL; DR

- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!

HPSS

https://docs.nersc.gov/filesystems/archive/

HPSS Is a Tape System

- All data in HPSS eventually ends up on tape
  - Transfers in go first to disk cache, so they are very quick
- Tape is linear media
  - Data cannot be written anywhere, only appended at end
  - Reading and writing are sequential, not random-access
  - Robot must fetch tape, load it into drive, read forwards until file is reached, then read file
    - Number-of-files has bigger impact on access performance than number-of-GB
- If you are retrieving more than ~100 files, please order your retrievals by tape position
  - NERSC has a helper script and instructions to help you sort

HPSS Best Practices

- Store files as you intend to extract them
  - Backup to protect against accidental deletion: use htar to bundle up each directory
  - Archiving data mirror: bundle by month data was taken or detector run characteristics, etc.
- Optimal size of bundles is currently 100s of GB
  - Larger than >1 TB retrieval is prone to interruption
- User xfer queue for long running transfer
  - Archiving during compute job only gets single stream data movement AND costs allocation hours
- Make sure your data gets archived
  - Collect log info when running htar AND check it for error reports (especially before deleting original data)
  - can also calculate checksums after the fact or enable during transfer

Checking HPSS Usage

On iris.nersc.gov

nstaff | CPU | GPU | Jobs | Storage | Roles | Groups | Details | History

Rows per page: 30 | Page 1 of 2 | Go to: 1 | .csv

HPSS Archive

HPSS Allocation: 16.6 PB

HPSS Storage Used: 19.0 PB

HPSS Storage Remaining: 0.0 B

% HPSS Storage Used: 114.6%

Allocation transfer report

Update HPSS Allocations

User	Username	User Quota	% Used	Storage Used	% Allowed by Project
		16.6 PB	99.1%	16.5 PB	100
		16.6 PB	4.5%	768.2 TB	100
		1.7 PB	26.5%	450.2 TB	10
		16.6 PB	2.2%	373.0 TB	100
		8.3 PB	4.3%	363.1 TB	50
		16.6 PB	1.1%	187.2 TB	100
		16.6 PB	1.0%	170.0 TB	100
		16.6 PB	0.3%	55.3 TB	100
		3.3 PB	1.1%	38.2 TB	20
		16.6 PB	0.2%	36.7 TB	100
		1.7 PB	2.0%	34.5 TB	10
		3.3 PB	1.0%	32.5 TB	20
		1.7 PB	1.5%	24.9 TB	10
		1.7 PB	1.2%	20.5 TB	10

Checking HPSS Usage

On the command line with “showquota”:

```
perlmutter login05> showquota --hpss
+-----------------------------------------+------------+-------------+------------------+------------+-------------+------------------+
|                             File system | Space used | Space quota | Space used (%)   | Inode used | Inode quota | Inode used (%)   |
+-----------------------------------------+------------+-------------+------------------+------------+-------------+------------------+
|                                    home |   28.94GiB |    40.00GiB |            72.4% |    369.27K |       1.00M |            36.9% |
|                                pscratch |    1.21TiB |     1.00PiB |             0.1% |      3.83M |      10.00M |            38.3% |
|   lgerhard usage on HPSS charged to nstaff |   38.15TiB |     3.32PiB |             1.1% |          - |           - |                - |
|  lgerhard usage on HPSS charged to nvendor |      0.00B |           - |                - |          - |           - |                - |
|    lgerhard usage on HPSS charged to m1093 |      0.00B |     8.30PiB |             0.0% |          - |           - |                - |
|  lgerhard usage on HPSS charged to dasrepo |      0.00B |    47.00TiB |             0.0% |          - |           - |                - |
|  lgerhard usage on HPSS charged to gpuover |      0.00B |     1.00TiB |             0.0% |          - |           - |                - |
+-----------------------------------------+------------+-------------+------------------+------------+-------------+------------------+
```

TL; DR

- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!

$HOME

- All-flash filesystem (fast access)
- Small block size (good for small files - source code, scripts, etc)
- Backed up
  - Daily snapshots
  - e.g. my homedir is at `/global/homes/e/elvis/.snapshots/2024-02-19`
  - (note: you can't see `.snapshots` with `ls`, but you can `cd` to it)
  - Also backed up to tape approximately monthly

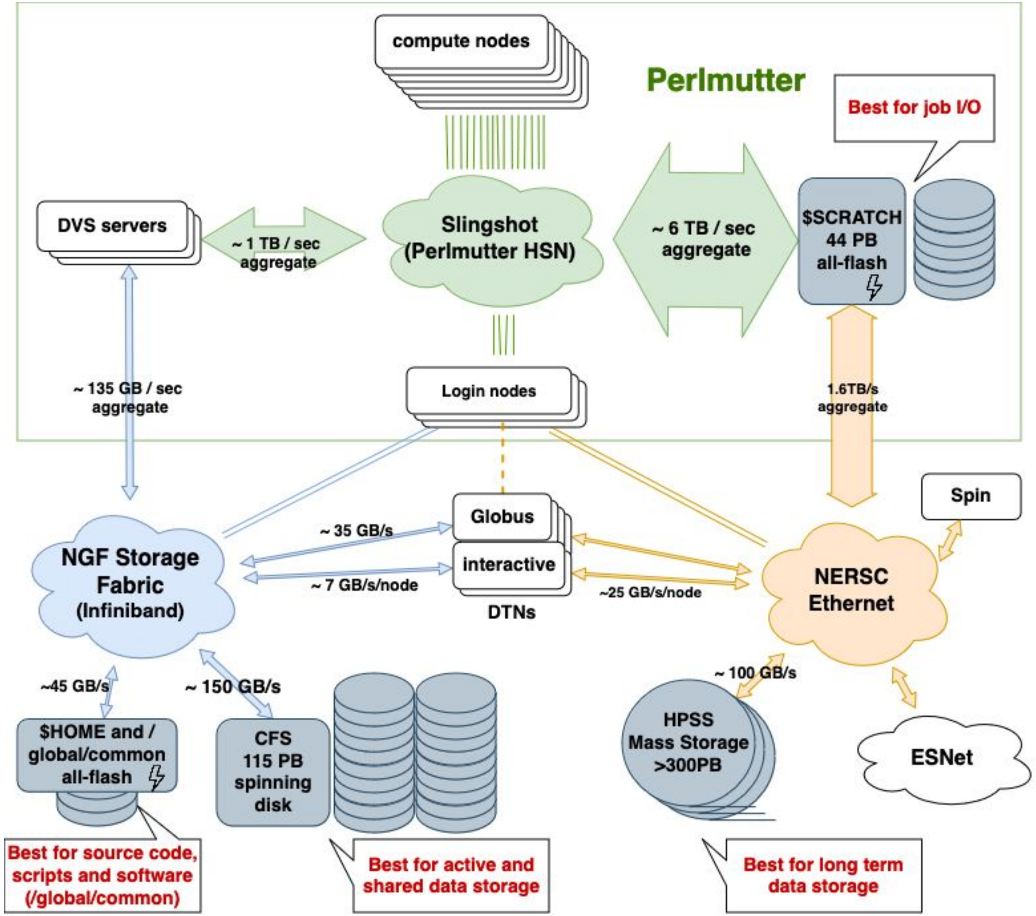
- Not for large I/O (relatively lower bandwidth)
- Small - not intended for data storage
- Not suitable for running jobs against
- Avoid making your conda environments here, particularly if you will use them in compute jobs!

TL; DR

- (Almost) All job I/O should happen on $SCRATCH
- Don't do I/O at scale on CFS
- CFS is best for actively-used data (but not source code)
- Put source code and conda environments in /global/common/software (or better still, a container)
- Not using it for a while? Bundle it into big-ish (100GB->2TB) tar files and store on HPSS
- $HOME only for small scale tests
- Have an off-NERSC copy of everything important!

Manage your data!

- **It's every user's responsibility to manage their data!**
- Different storage systems are optimized for different parts of the data lifecycle
- NERSC provides tools to help you manage your data (docs.nersc.gov has a "Managing Data" section to help you use them)



Q&A