OAK RIDGE National Laboratory

# Node Local Storage: Common Use Cases and Some Tools to Help

Christopher Zimmer

Mike Brim (UnifyFS)

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

# Summit Storage Perspective

- Primary Storage: Alpine:
  - 250 PB GPFS Parallel File System
  - 2.5 TB/s read and write
  - Order of 100,000,000 IOPS

- Specialized Storage: Node Local NVMes

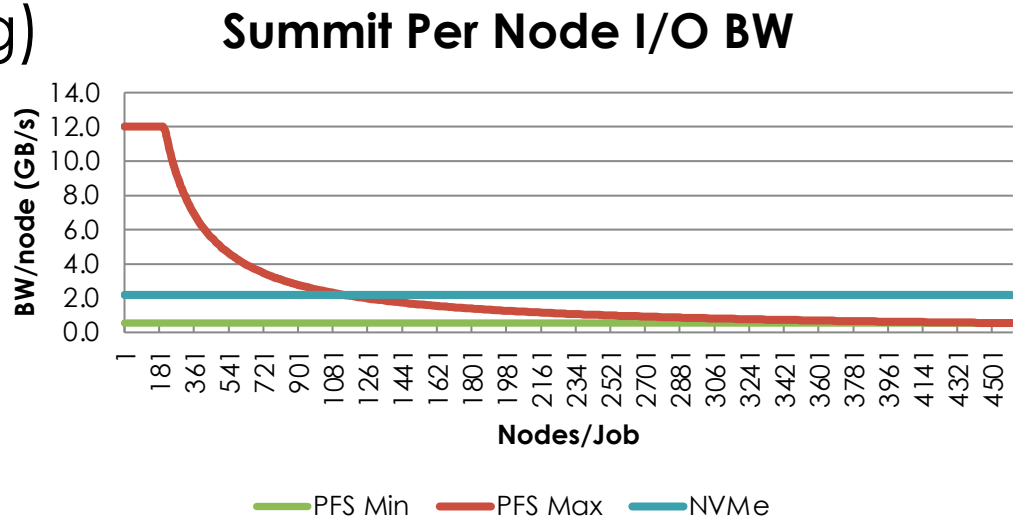
  -Using all nodes
    - 7.3 PB
    - 25 TB/s Read
    - 9.6 TB/s Write
    - > 1,000,000,000 IOPS

# Fronter Storage Perspective (Public)

- Primary Storage: Orion
  - ~679PB Lustre Multi-Tier Parallel File System
  - 10 TB/s Distributed Flash Tier
  - ~5 TB/s Capacity Tier
  - Order of 100,000,000 IOPS
- Specialized Storage: Node Local NVMes
  - >10 PB Capacity
  - >60 TB/s Read
  - >25 TB/s Write
  - >10B IOPs

# When to use the Burst Buffers (Node Scale)

- Large block/Small-Medium sized streaming inputs/outputs on periodic basis may benefit most from parallel file-system
- Small granularity access or exceptionally large I/O needs best served by node-local
  - >50TB/hour read/write  (Checkpointing)
  - <32K high frequency access (Training)



# Checkpointing/Output

- When to consider:
  - I/O overheads > 5% - What does this look like?
    - Recent analysis of workload at ~100TB/hour
    - 3-5 minutes per checkpoint (why) shared access, capturing full performance generally requires ideal access patterns
    - 15-20 seconds checkpoints to NVMe (drain in the background)
  - 72 minutes of I/O reduced to 8 minutes.
- Frontier will be a larger machine with more transistors, checkpoint frequency will like need to increase.

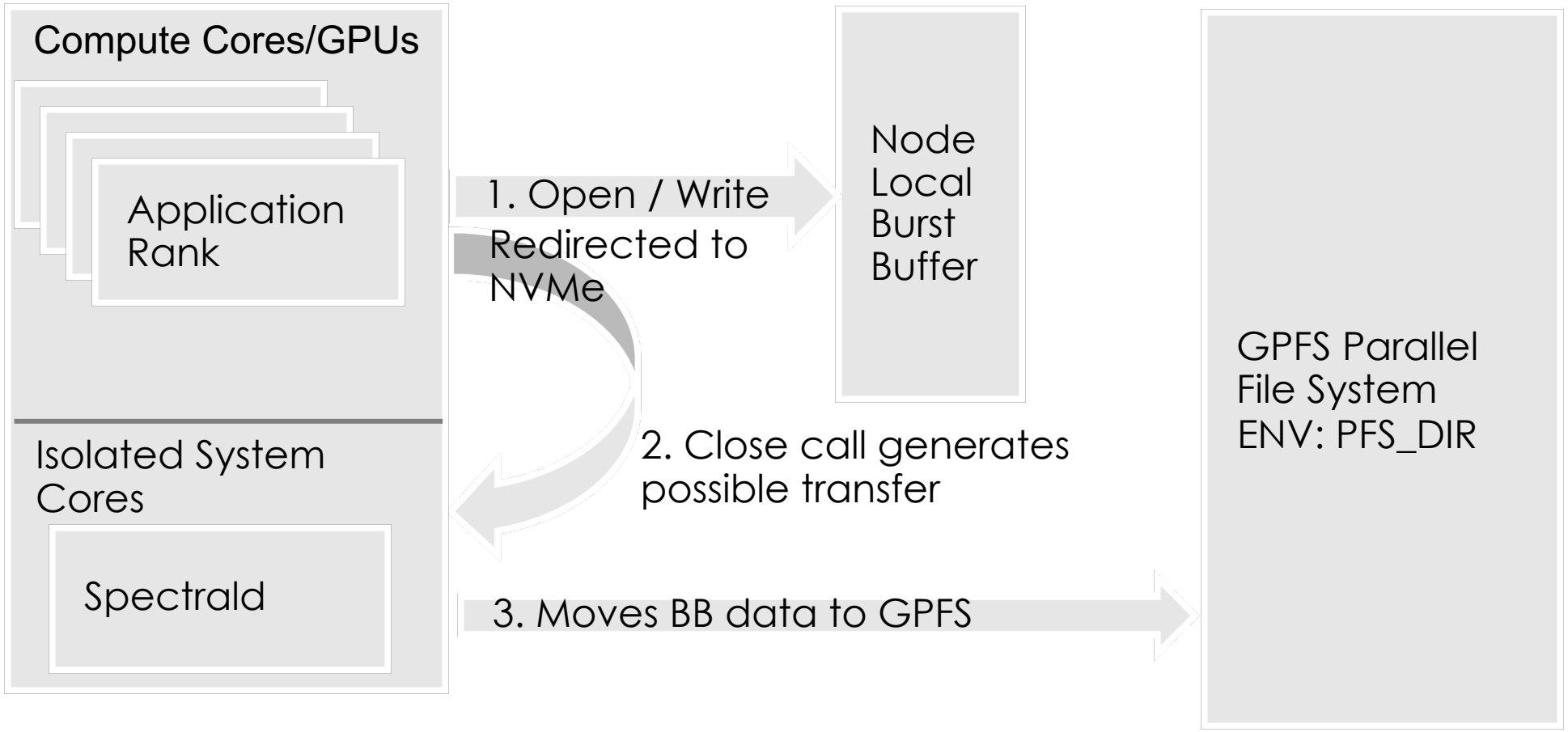
# Manual: Grab an NVMe and Go

- Allocate node local storage
- bsub -Is -nnodes 1 -PSTF008 -W00:10 -alloc_flags "nvme" /bin/bash
  - jsrun -r1 df
    - /dev/mapper/bb-bb1 1452706772 33040 1452673732 1% /mnt/bb/cjzimmer
- Batch:
  - #!/bin/bash -l
  - #BSUB -P STF008
  - #BSUB -W 01:00
  - #BSUB -nnodes 1
  - #BSUB -alloc_flags "gpumps smt4 **nvme**"

# Spectral

- On node copy agent
  - Runs on isolated cores as system agent
- Application Interface Transparent
  - No code modifications (LD_PRELOAD)
    - Changes limited job scripts
  - Application only reasons about a single namespace
- Preserves portability with single namespace

# Spectral Data Flow



# Spectral errata

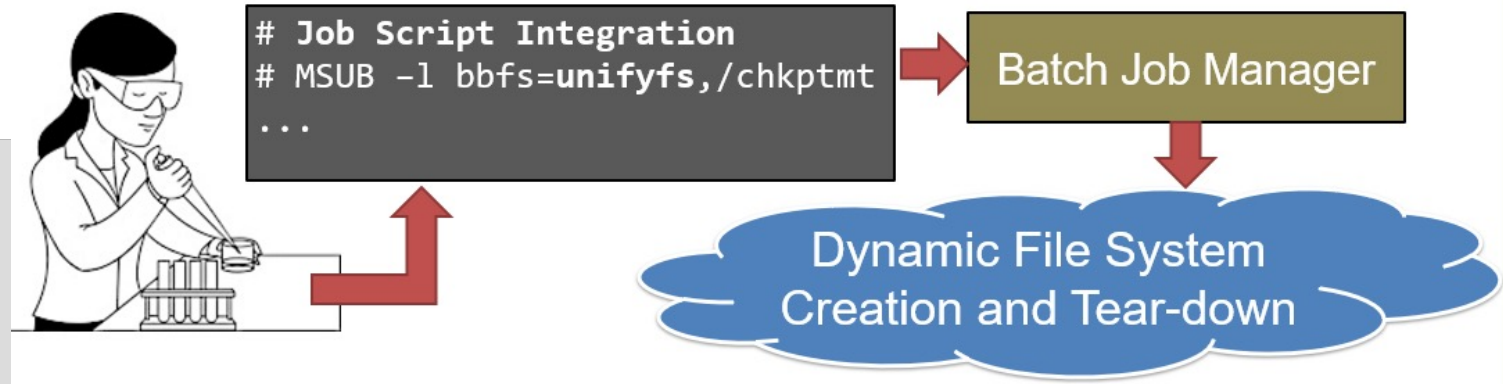
- Spectral tracks no meta data
  - Sophisticated I/O patterns including hazards (RAW, WAW) will result in corruption
    - The trade off is speed, meta data is expensive, raw performance can be achieved

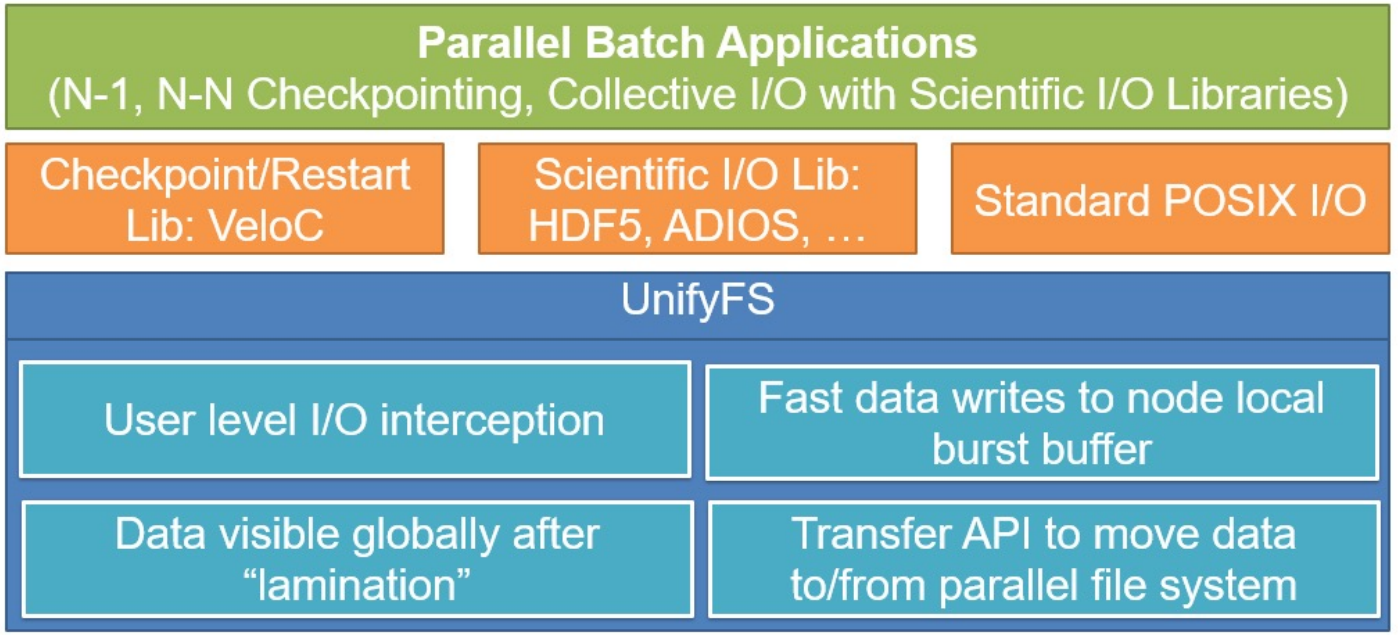
OAK RIDGE National Laboratory

EXASCALE COMPUTING PROJECT

# UnifyFS

- Job-duration user-level file system
- Provides *unified* namespace over independent node-local storage
- No application code changes required, and minimal job script changes




Source Code on GitHub:
https://github.com/LLNL/UnifyFS

Online documentation:
https://unifyfs.readthedocs.io/en/latest/index.html

Lawrence Livermore National Laboratory

NCSA

OAK RIDGE National Laboratory

# UnifyFS Overview

- Designed for efficient use of node-local storage for applications with typical HPC bulk-synchronous I/O patterns
  - supports both shared files (N-1) and file-per-process (N-N)
- Transparently intercepts I/O: POSIX, MPI-IO, HDF5
- Usage: 3 Easy Steps
  1. Update application file paths to use prefix `/unifyfs`
     - often, this does not require code changes
  2. Link application with the UnifyFS client library
  3. Update job script to start/stop UnifyFS servers at beginning/end of job (with optional stage-in and stage-out)

# UnifyFS File Sharing Semantics

- Application data written to node-local storage is only visible to remote application processes after an explicit sync operation
  - POSIX I/O: `fsync()` or `close()`
  - MPI-IO: `MPI_File_sync()`
  - HDF5: `H5Fflush()`

- File Lamination: mark file as read-only, no further writes allowed
  - allows UnifyFS to optimize metadata management for faster reads
  - to laminate, use UnifyFS API or `chmod(0444)`

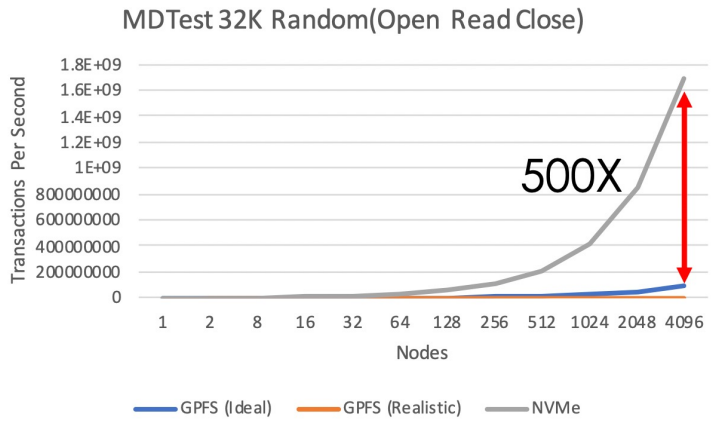
# UnifyFS - Summit Quickstart Guide

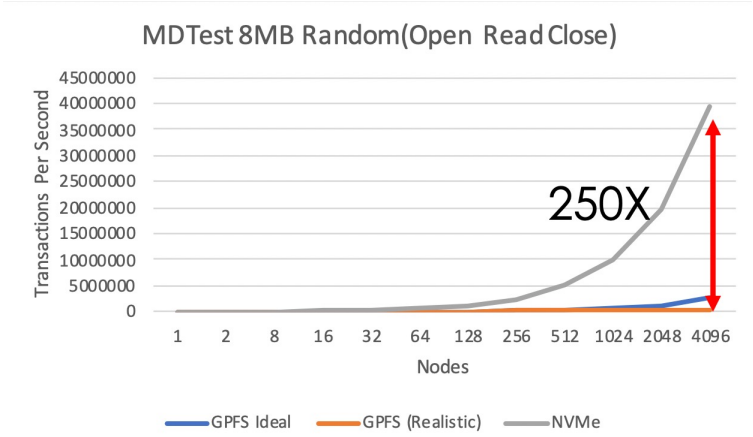
- `module use /sw/summit/unifyfs/modulefiles`
- `module avail unifyfs/1.0-beta`
- `module load unifyfs/1.0-beta/mpi-mount-<compiler>`
- `unifyfs-help # for info on job integration and configuration`

- For Help/Issues/Suggestions:
  - Do **not** contact OLCF help
  - Send an email to ecp-unifyfs@exascaleproject.org

# Machine Learning

- Training in particular stresses the file-system
  - Lots of small file reads and re-reads put tremendous pressure on the system.



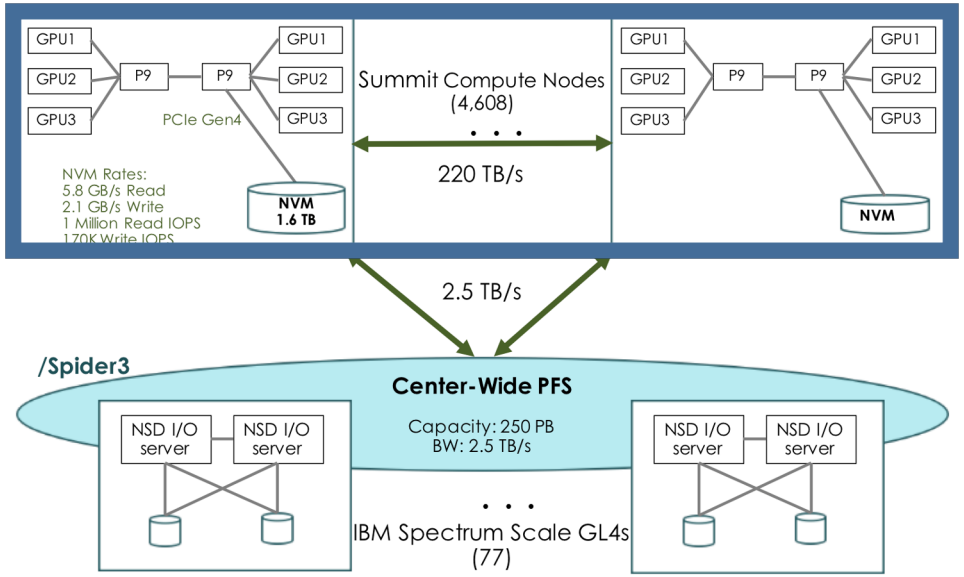



# Common Technique

- Sharded Training
  - Manual process
    - 1. Cut dataset into shards that fit on NVMe
    - 2. Copy shards to NVMe prior to run
    - 3. Train on data from NVMe
  - While this is tedious, it's shown significant performance benefit
  - Can impact convergence

# New Tool (Summer 2022)

- Eliminates sharding
  - Intercepts reads and caches them locally on NVMe
  - Subsequent reads within the same application run fulfill subsequent accesses without PFS.
  - Eliminates meta data overhead
  - Increases BW availability



# Pre-Question

- How one would set up using a parallel MPI write across multiple nodes that could be collected with the Burst Buffer?
  - Spectral or UnifyFS are tools that can do this
  - Chimera-IO writes 10 HDF5/MPIIO Files
    - Spectral redirects the individual rank portions to the NVMe
    - On a 96 node test case
      - I/O timing is cut from 353s to 135s
        - @96 GPFS has more aggregate BW than 96 NVMe