

Accelerating application

I/O with unify FS

ECP Community BOF - DAVS
March 30, 2021



LLNL: KATHRYN MOHROR (PI), ADAM MOODY, CAMERON STANAVIGE, TONY HUTTER

ORNL: SARP ORAL (ORNL-PI), HYOGI SIM, FEIYI WANG, MIKE BRIM, SEUNG-HWAN LIM, JENNA DELOZIER

NCSA: CELSO MENDES, CRAIG STEFFEN



What is UnifyFS?

- Simply put, it's a file system for burst buffers
- Our goal is to make using burst buffers on exascale systems as *easy* as writing to the parallel file system and orders of magnitude *faster*

```
int main(int argc, char **argv) {
    MPI_Init(argc, argv);

    for (t = 0; t < TIMESTEPS; t++) {
        /* do work ... */
        checkpoint();
    }

    MPI_Finalize();

    return 0;
}
```

```
void checkpoint(void) {
    int rank;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // file = "/pfs/shared.chpt";
    file = "/unifyfs/shared.ckpt";

    File *fs = fopen(file, "w");

    if (rank == 0)
        fwrite(header, ..., fs);

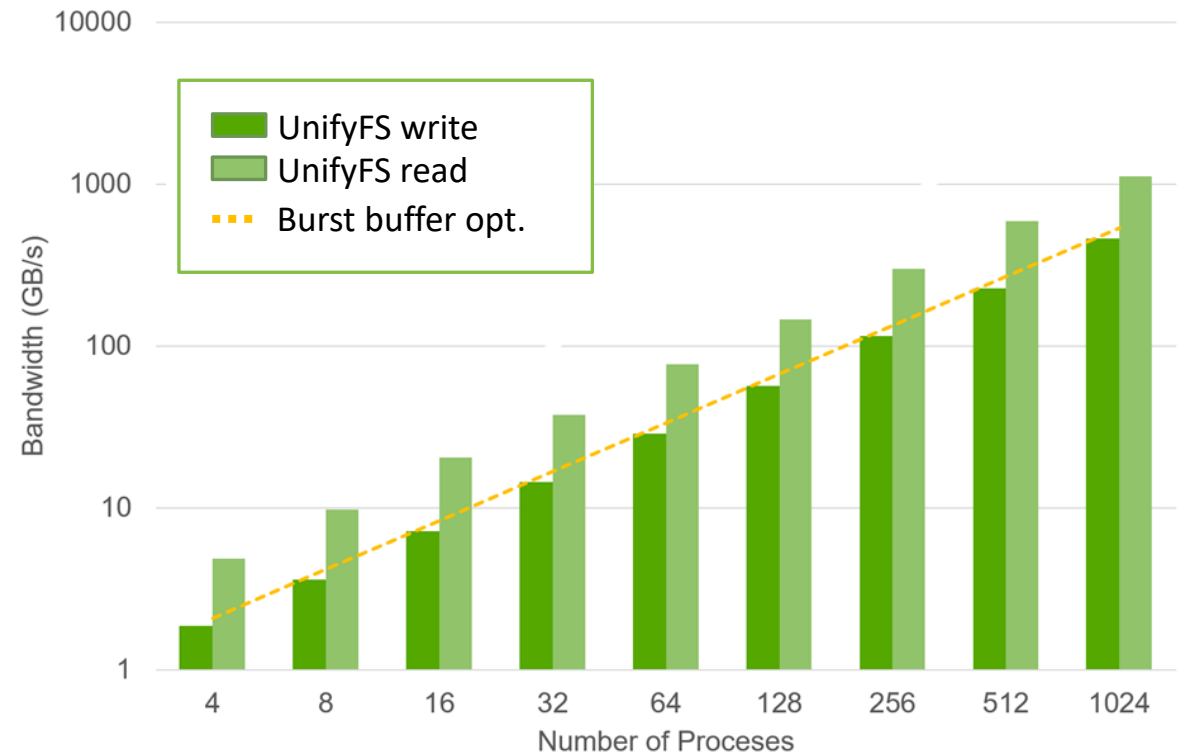
    long offset = header_size +
                  rank*state_size;
    fseek(fs, offset, SEEK_SET);
    fwrite(state, ..., fs);
    fclose(fs);
}
```

The only required change is to use **/unifyfs** instead of /pfs



UnifyFS targets local burst buffers because they are fast and scalable

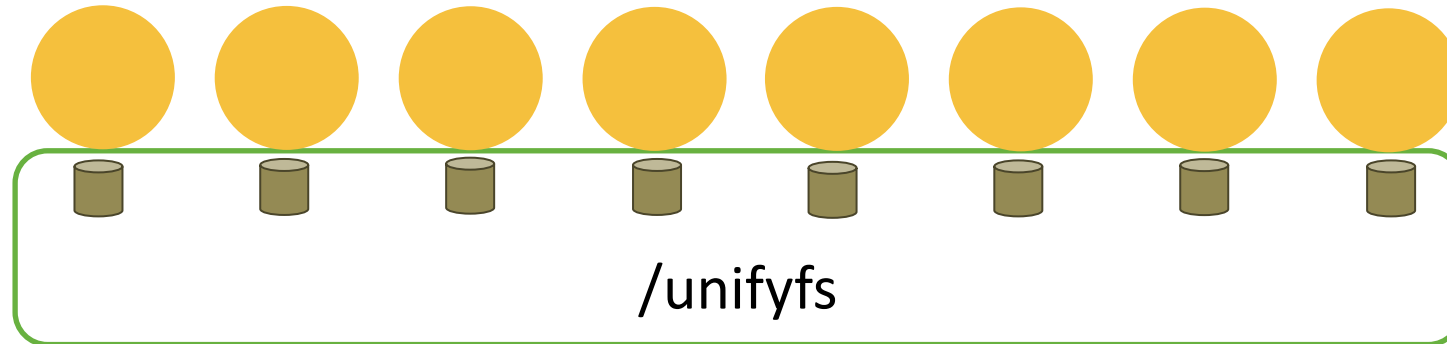
- UnifyFS (v0.9.1) scaling tests on Summit
 - All writes using burst buffer (no file data stored in memory for these runs)
 - Write performance is equal to the cumulative theoretical throughput of the node-local burst buffer
 - Read performance takes advantage of caching





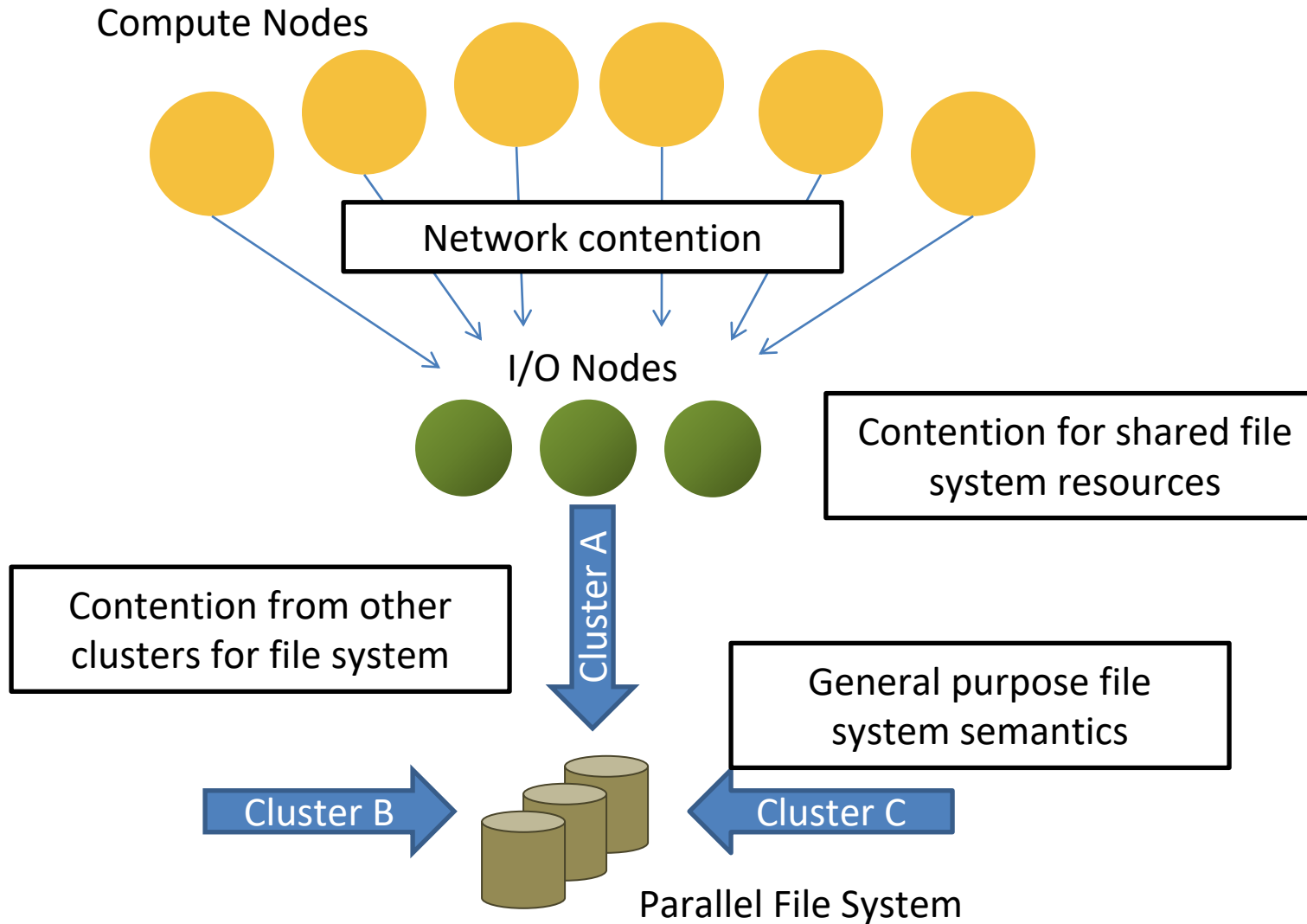
UnifyFS makes sharing files on node-local burst buffers easy and fast

- Sharing files on node-local burst buffers is not natively supported



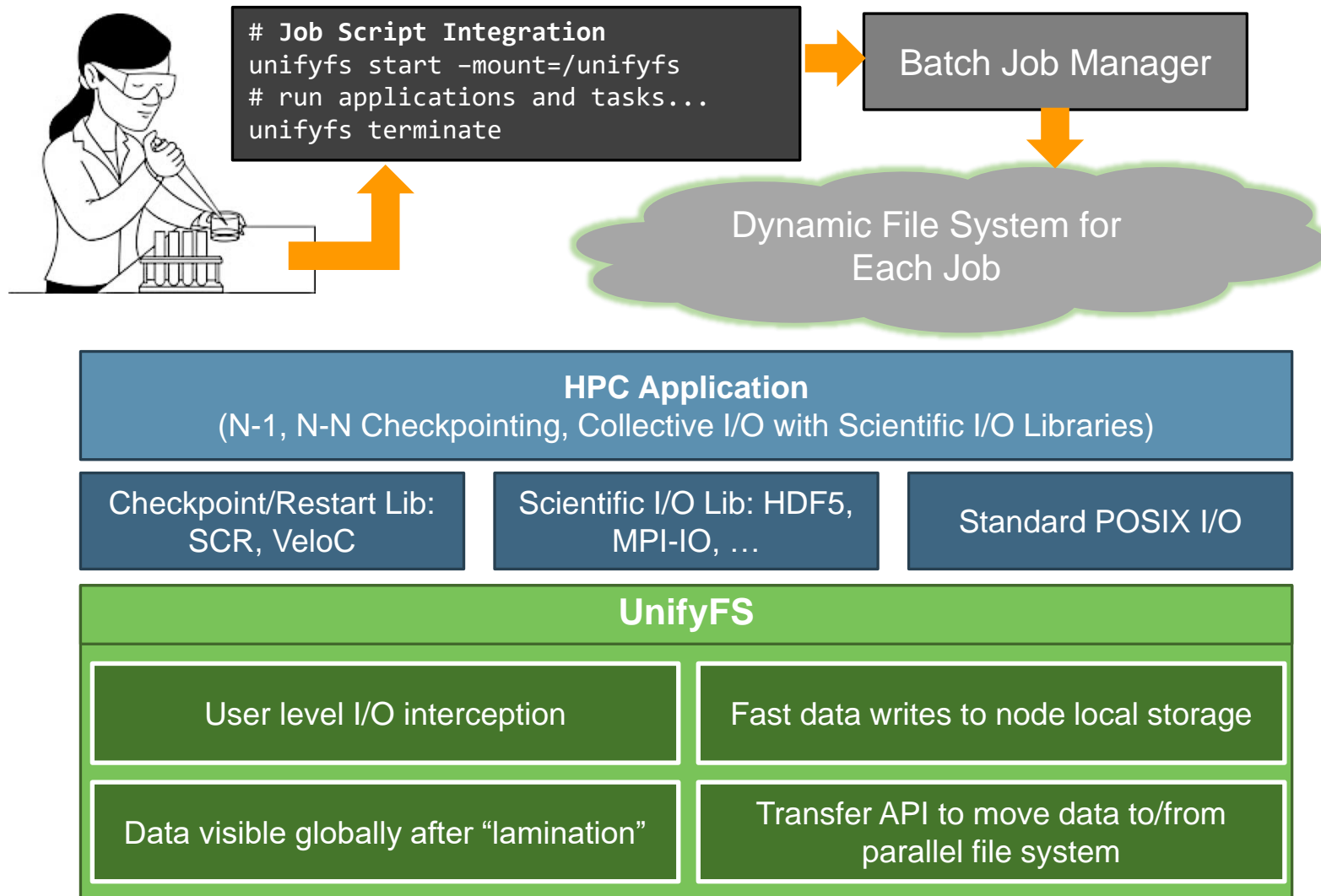
- UnifyFS makes sharing files **easy**
 - UnifyFS presents a shared namespace across distributed storage
 - Used directly by applications or indirectly via higher level libraries like VeloC, MPI-IO, HDF5, PnetCDF, ADIOS, etc.
- UnifyFS is **fast**
 - Tailored for specific HPC workloads, e.g., checkpoint/restart, visualization output, loose coupling through files
 - Each UnifyFS instance exists only within a single job, no contention with other jobs on the system

Writing data to the parallel file system is expensive





UnifyFS is designed to work completely in user space for a single user's job





We need you!

- Our goal is to provide **easy, portable, and fast** support for burst buffers for ECP applications
- We need early users
- What features are most important to you
- Available on github:
<https://github.com/LLNL/UnifyFS>
- MIT license
- Documentation and user support
 - User Guide: <http://unifyfs.readthedocs.io>
 - unifyfs@exascaleproject.org

UnifyFS: A file system for burst buffers

User Guide

- Overview
 - High Level Design
- Definitions
 - Job
 - Run or Job Step
- Assumptions
 - Application Behavior
 - Consistency Model
 - File System Behavior
 - System Characteristics
- Build & I/O Interception
 - UnifyFS Build Configuration Options
 - How to Build UnifyFS
 - I/O Interception
- Mounting UnifyFS
 - Mounting
 - Unmounting
- UnifyFS Configuration
 - unifyfs.conf
 - Environment Variables
 - Command Line Options
- Starting & Stopping in a Job
 - Starting UnifyFS
 - Stopping UnifyFS
 - Process Manager

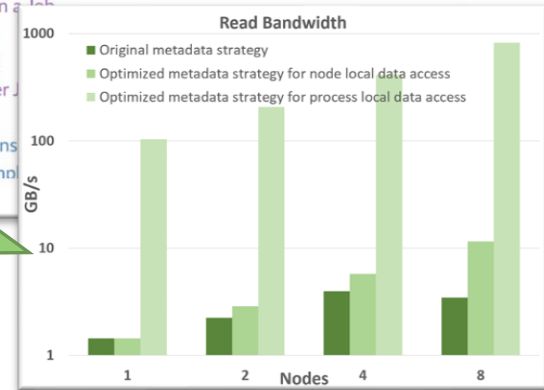
It's EASY!

```
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    ...
    for (t = 0; t < TIMESTEPS; t++) {
        /* do work ... */
        checkpoint();
    }
    MPI_Finalize();
    return 0;
}

void checkpoint(void) {
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // file = "/pfs/shared.chkpt";
    file = "/unifyfs/shared.ckpt";
    File *fs = fopen(file, "w");
    if (rank == 0)
        fwrite(header, ..., fs);
    long offset = header_size +
        rank*state_size;
    fseek(fs, offset, SEEK_SET);
    fwrite(state, ..., fs);
    fclose(fs);
}
```

The only required change is to use `/unifyfs` instead of `/pfs`

It's FAST!





This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.



This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.