Emulating I/O Behavior in Scientific Workflows on High Performance Computing Systems

Fahim Tahmid Chowdhury*, Yue Zhu*, Francesco Di Natale+, Adam Moody+, Elsa Gonsiorowski+, Kathryn Mohror+, Weikuan Yu*

Florida State University*
Lawrence Livermore National Laboratory+

PDSW 2020

Outline

- **Understanding HPC Workflow I/O**
- Wemul: HPC Workflow I/O Emulation Framework
- Experimental Results
- Future Work

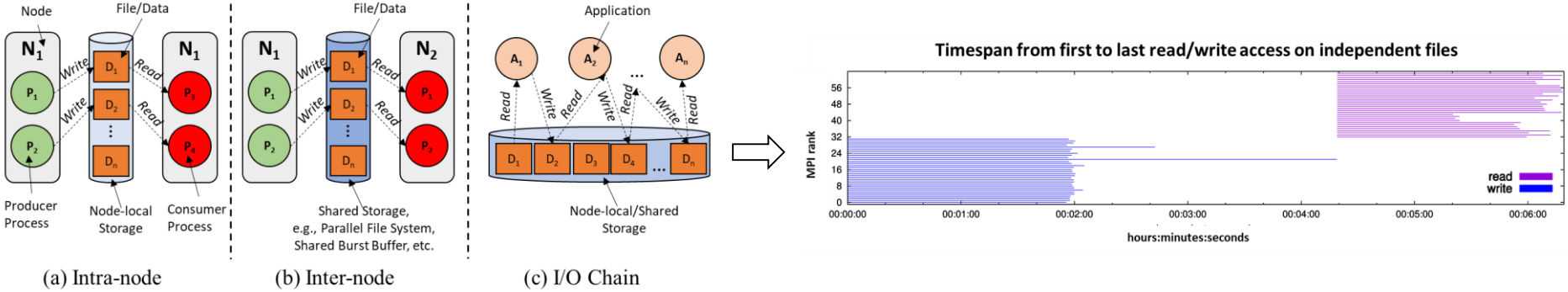
HPC Workflow and Dataflow

- What is HPC Workflow?
  - Pre-defined or random ordered execution of set of tasks
  - Target can be achieved by inter-dependent or independent applications

- Scientific applications on HPC can create complex workflows
  - Managing multi-scale simulations, e.g., high-energy physics, material science and biological science, etc.
  - Coupling multi-physics codes, e.g., climate models
  - Cognitive simulations and ensembles, e.g., optimization and uncertainty quantification

- Dataflow or data transfer in HPC Workflows can create bottlenecks due to data-dependency among workflow modules

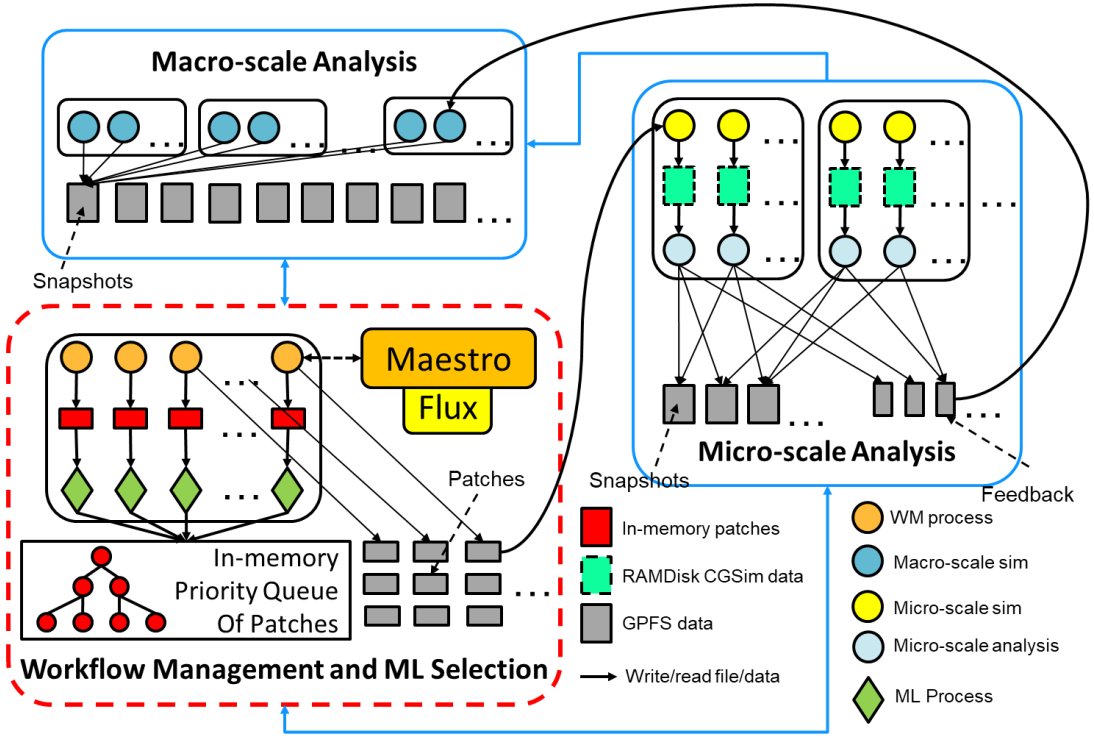
Simple Workflow: Producer-Consumer I/O



(a) Intra-node (b) Inter-node (c) I/O Chain

- Producer and consumer processes can reside on same or different nodes
- Inter-node producer-consumer processes need shared resource for data transfer
- Contention among tasks for shared resource can hinder the overall performance

Complex Workflow: Cancer Moonshot Pilot-2



- Simulation of RAS protein and cell membrane interaction to help early stage cancer diagnosis
- Run by Multiscale Machine-Learned Modeling Infrastructure (MuMMI)[1]
- 4K Sierra nodes with 16K GPUs and 176K CPU cores
- Macro-scale analysis generates 400M files of over 1PB total size

[1] F. Di Natale et al., "A Massively Parallel Infrastructure for Adaptive Multiscale Simulations: Modeling RAS Initiation Pathway for Cancer", SC'19

HPC Workflow I/O Challenges

- Scale and complexity pose significant challenges
  - Coupling diverse types of applications
  - Handling failures
  - Scheduling millions of tasks on compute
  - Managing humongous amount of data using cutting-edge storage stack

- Understanding I/O behavior from workflow perspective is a pre-requisite to data management strategy development
  - Challenge 1: Scarcity of actual workflow source code
  - Challenge 2: Tight dependency of workflow on specific supercomputing cluster
  - **Solution: System-agnostic framework to emulate HPC workflow I/O workloads**

Existing I/O Analysis Tools

- Synthetic Benchmarks
  - IOR, IOZone, FIO, Filebench, etc.
  - Limitation: Difficult to closely mimic real application behavior
- Application Benchmarks
  - CM1, Montage, HACC I/O, VPIC I/O, FLASH3 I/O, etc.
  - Limitation: Non-generic application-specific tools
- I/O workload modeling and simulation tools
  - IOWA, MACSio, etc.
  - Limitation: Not possible to address data dependency among the workflow tasks

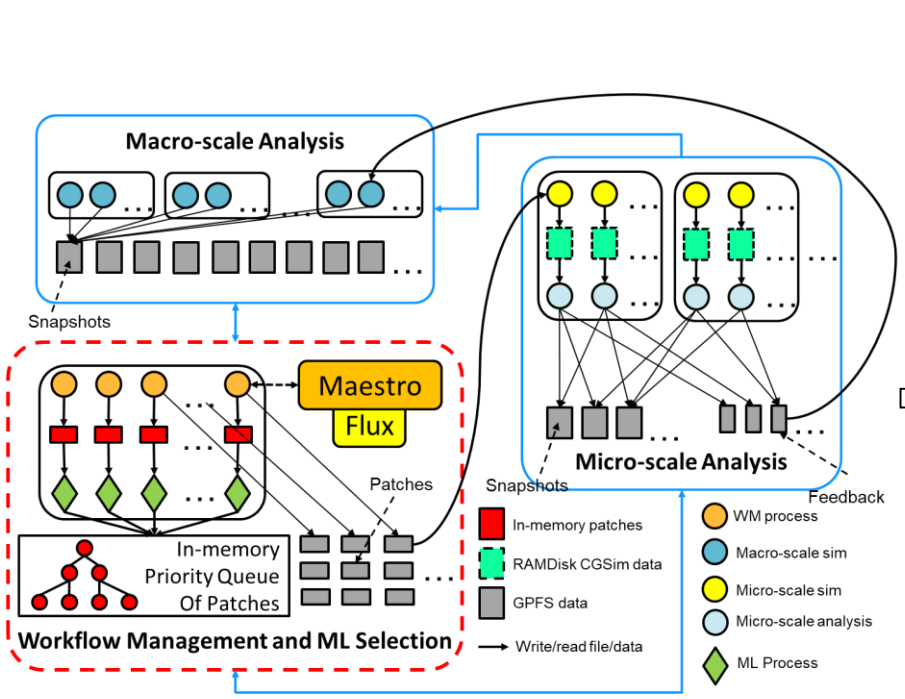
Important Research Questions

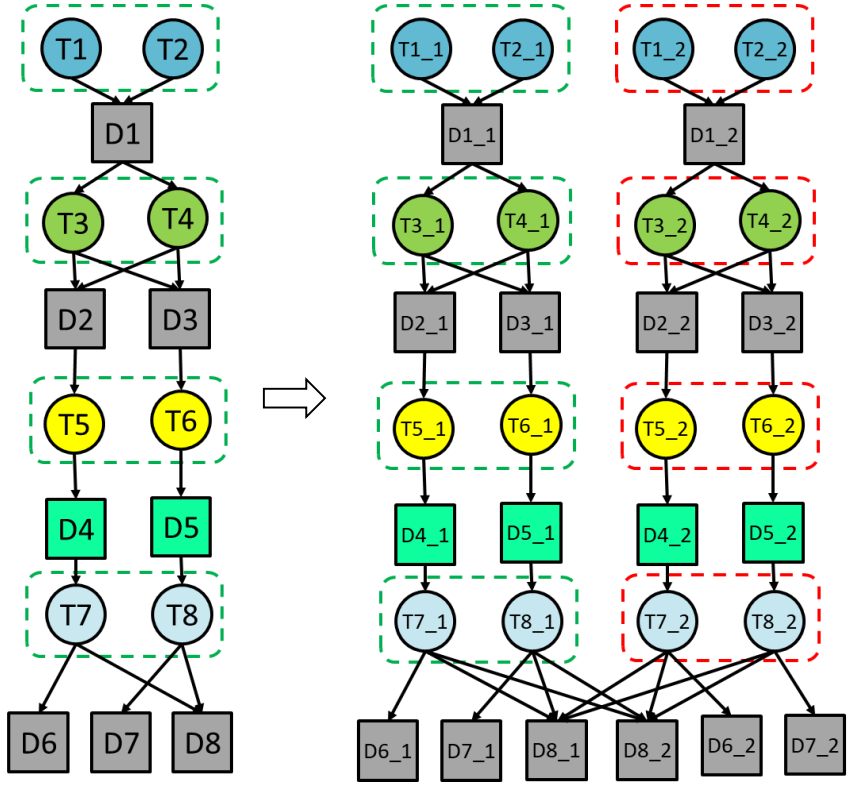
- How to address the **data-dependency** among workflow modules?
- How to mimic generic **complex workflow** with/without cycles?
- How to develop a **system-agnostic** emulation framework?
- How to leverage the framework for workflow **workload analysis**?

Outline

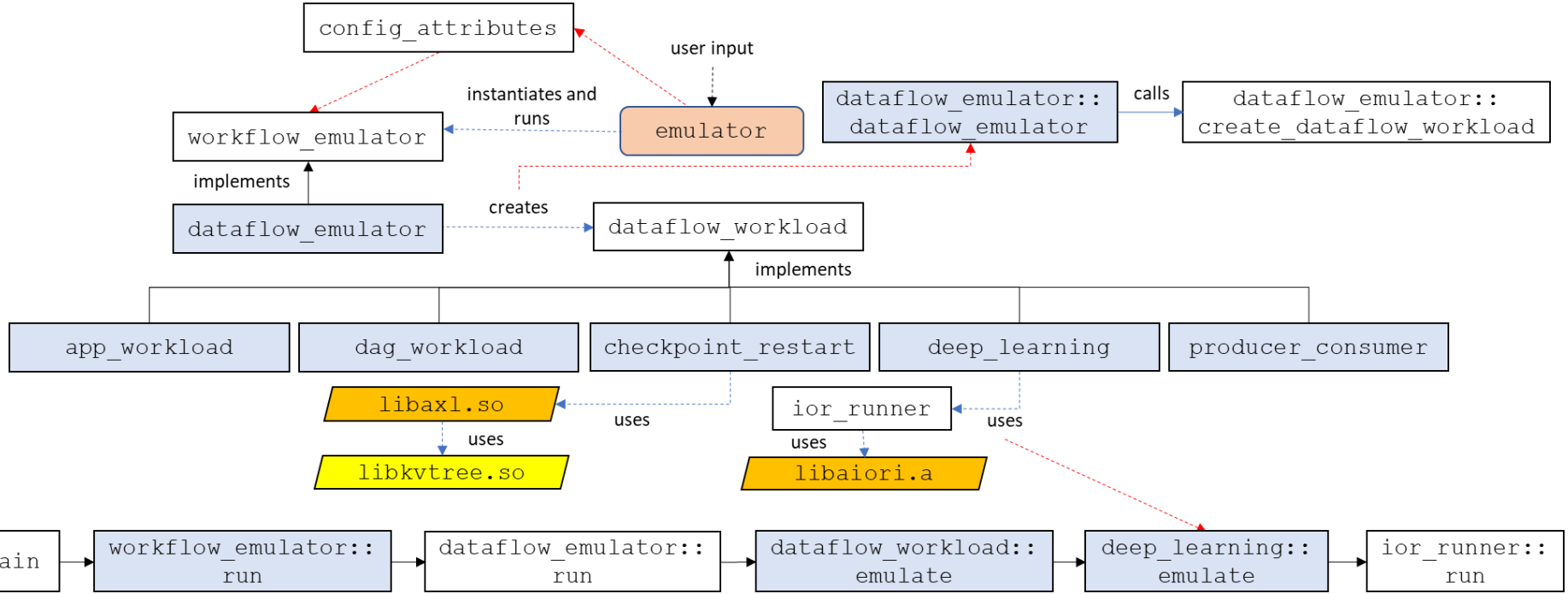
- Understanding HPC Workflow I/O
- **Wemul: HPC Workflow I/O Emulation Framework**
- Experimental Results
- Future Work

Graph Representation of Data-dependency





Wemul: Software Architecture



Wemul: Execution Modes

- DL training
  - Recursively traverse all files in a dataset directory and equally assign to each process
  - Read all files in parallel

Parameter	Description
`--input_dir <path>`	Mountpoint or path to storage system to use
`--block_size <size in bytes>`	Block size per read or write request
`--segment_count <number>`	Total number of blocks or segments
`--use_ior (optional)`	Enable using IOR as a library
`--num_epochs <number>`	Number of epochs in DL training experiment
`--comp_time_per_epoch <time in seconds>`	Computation emulation per epoch

Wemul: Execution Modes (contd.)

- Producer-consumer
  - Inter- or intra-node modes
  - Can be run as standalone producer or consumer, but not both

Parameter	Description
`--inter_node`	Set for enabling inter-node producer-consumer
`--producer_only`	Run Wemul as standalone producer application
`--consumer_only`	Run Wemul as standalone consumer application
`--ranks_per_node <number>`	Feed ranks per node number to help intra- or inter-node data transfer

Wemul: Execution Modes (contd.)

- Application-based
  - Run Wemul as a standalone application
  - Set the list of files to read/write and a list of mount point paths
  - Set block size, segment count and access pattern, i.e., file-per-process or shared-file

Parameter	Description
`--read_input_dirs <dir1:dir2:..>`	Colon separated list of mountpoints to storage systems for reading
`--read_filenames <file1:file2:..>`	Colon separated list of files to be read
`--read_block_size <size in bytes>`	Block size for the files to be read
`--read_segment_count <number>`	Segment count for the files to be read
`--file_per_process_read`	Enable file-per-process read (shared read by default)
`--write_input_dirs <dir1:dir2:..>`	Colon separated list of mountpoints to storage systems for writing
`--write_filenames <file1:file2:..>`	Colon separated list of files to be written
`--write_block_size <size in bytes>`	Block size for the files to be written
`--write_segment_count <number>`	Segment count for the files to be written
`--file_per_process_write`	Enable file-per-process write (shared write by default)

Wemul: Execution Modes (contd.)

- DAG-based
  - Take graph representation of the entire workflow as input
  - Processes of the same application can have different access patterns
  - **`--dag_file <filepath>`**

```
APP A1 ml_process 1 12
APP A2 macro_an 2 8
APP A3 micro_sim 3 5
APP A4 micro_an 3 6
DATA D1 macro_snapshot 3
DATA D2 patch_1 3
DATA D3 patch_2 3
DATA D4 patch_3 3
DATA D5 sim_patch_1 3
DATA D6 sim_patch_2 3
DATA D7 sim_patch_3 3
DATA D8 feedback_1 3
DATA D9 feedback_2 3
DATA D10 feedback_3 3
DATA D11 micro_snapshot 3
```

```
PARENT A2 CHILD D1 ACCESS 0
PARENT D1 CHILD A1 ACCESS 0
PARENT A1 CHILD D2 D3 D4 ACCESS 0
PARENT D2 D3 D4 CHILD A3 ACCESS 1
PARENT A3 CHILD D5 D6 D7 ACCESS 1
PARENT D5 D6 D7 CHILD A4 ACCESS 1
PARENT A4 CHILD D8 D9 D10 ACCESS 1
PARENT A4 CHILD D11 ACCESS 0
NS_PARENT D8 D9 D10 NS_CHILD A2 ACCESS 0
```

```
TASK T1 ml_process 12
TASK T2 macro_1 8
TASK T3 macro_2 8
TASK T4 micro_sim_1 5
TASK T5 micro_sim_2 5
TASK T6 micro_sim_3 5
TASK T7 micro_an_1 6
TASK T8 micro_an_2 6
TASK T9 micro_an_3 6
DATA D1 macro_snapshot 3
DATA D2 patch_1 3
DATA D3 patch_2 3
DATA D4 patch_3 3
DATA D5 sim_patch_1 3
DATA D6 sim_patch_2 3
DATA D7 sim_patch_3 3
DATA D8 feedback_1 3
DATA D9 feedback_2 3
DATA D10 feedback_3 3
DATA D11 micro_snapshot 3
```

```
PARENT T2 T3 CHILD D1
PARENT D1 CHILD T1
PARENT T1 CHILD D2 D3 D4
PARENT D2 CHILD T4
PARENT D3 CHILD T5
PARENT D4 CHILD T6
PARENT T4 CHILD D5
PARENT T5 CHILD D6
PARENT T6 CHILD D7
PARENT D5 CHILD T7
PARENT D6 CHILD T8
PARENT D7 CHILD T9
PARENT T7 CHILD D8
PARENT T8 CHILD D9
PARENT T9 CHILD D10
NS_PARENT D8 D9 D10 NS_CHILD T2 T3
PARENT T7 T8 T9 CHILD D11
```

Outline

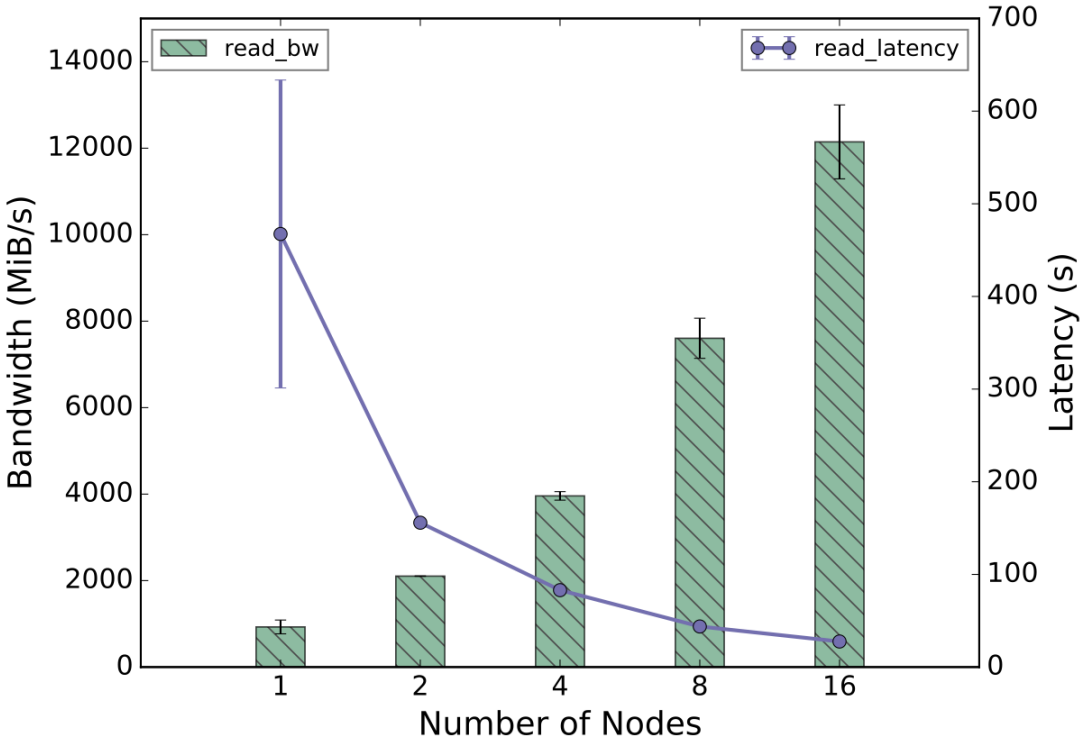
- Understanding HPC Workflow I/O
- Wemul: HPC Workflow I/O Emulation Framework
- **Experimental Results**
- Future Work

Experimental Setup

- HPC cluster: Lassen
  - IBM Power9 system 44 cores per node
  - 795 nodes
  - Memory: 256 GB per node
  - Parallel File System: 24 PB IBM Spectrum Scale (GPFS)
  - Burst Buffer: 1.6 TB on-node NVMe PCIe SSD devices per node
  - RAMDisk: 148 GB per node
  - tmpfs: 128 GB per node

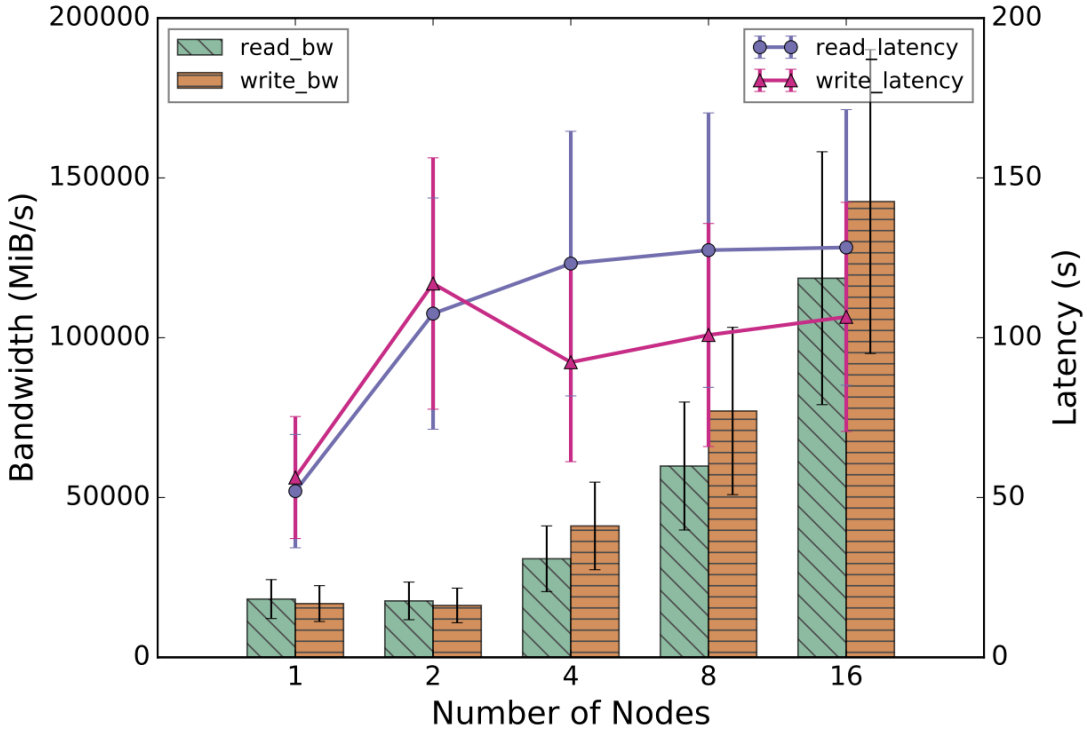
- Experiments on all execution modes using GPFS
  - 1 to 16 client nodes
  - 8 processes per node
  - Profiling Tool: Darshan-3.1.7

DL Training I/O on Lassen's GPFS



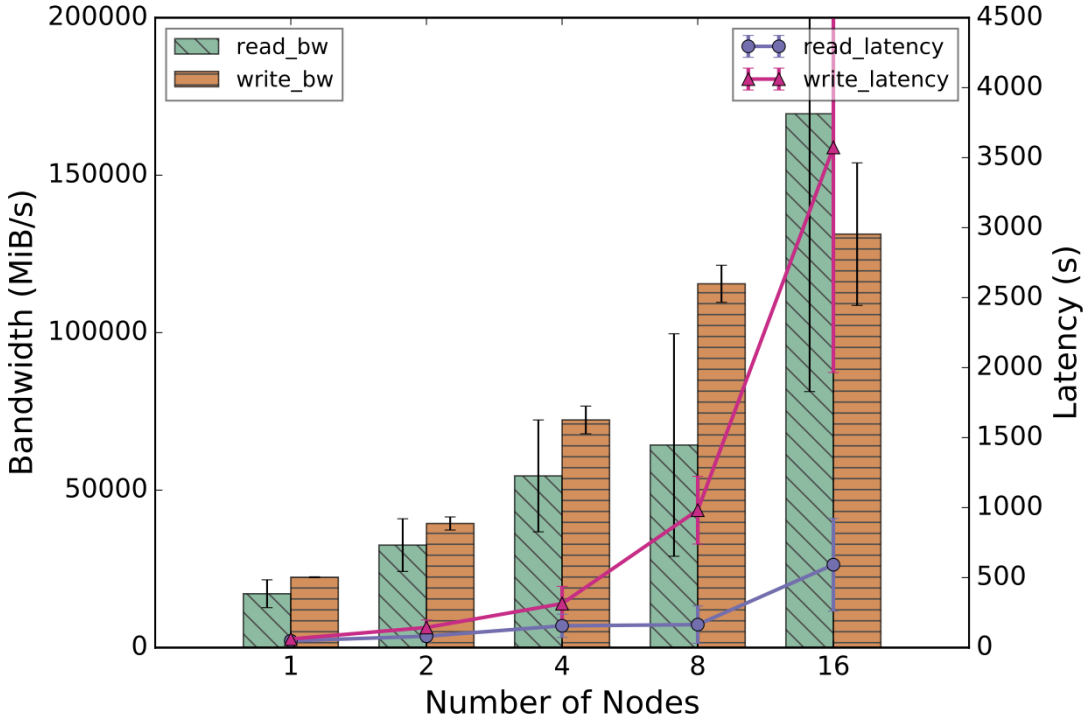
- Dataset: 327680 1 MiB files arranged equally in 320 subdirectories aggregating 320 GiB
- Emulate 3 epochs
- Run 5 times for each data point
- Reaches up to ~12 GiB/s read for 16 nodes and 8 processes per node
- Latency decreases with increasing processes, because each process has less files to read

Producer-Consumer I/O on Lassen's GPFS



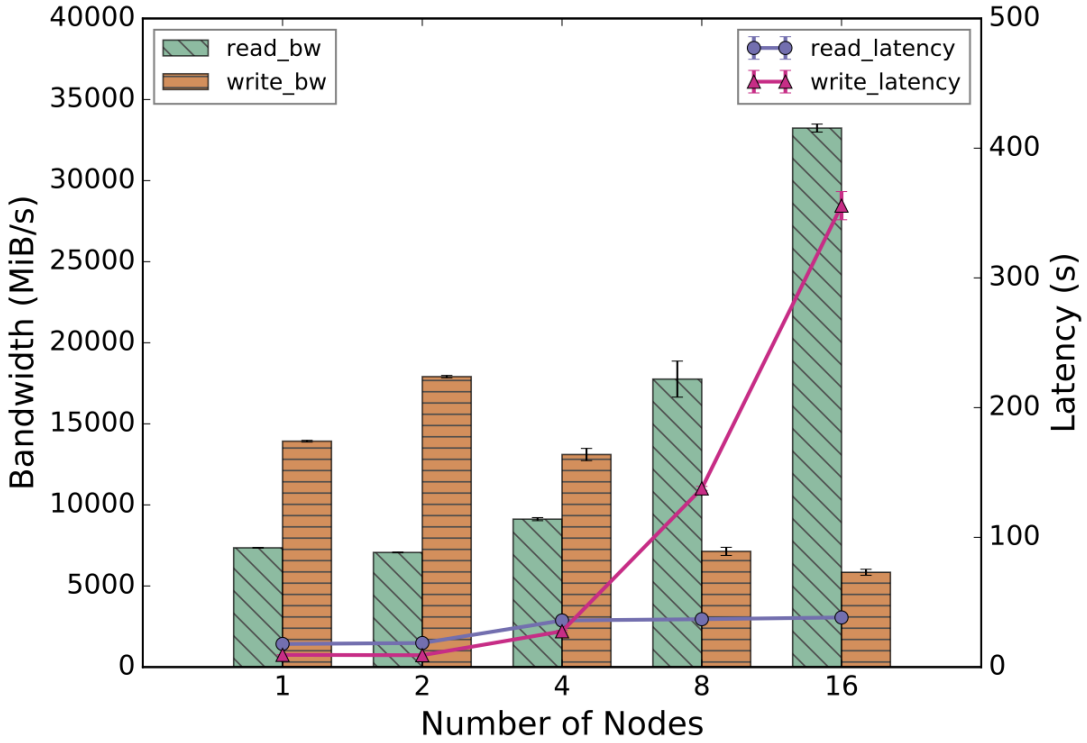
- Simple inter-node producer-consumer workflow
- 8 procs/node
- 32 G data produced by each process, and the same consumed by another
- ~2.2 TiB for 16 nodes
- Max ~118 GiB/s read b/w
- Max ~142 GiB/s write b/w

Application-based I/O on Lassen's GPFS



- 3 stage producer-consumer workflow
- ❑ Stage 1: Write #(procs/2) 32G files with shared access
- ❑ Stage 2: Read files from stage 1 with shared-access and write #(procs) 16G files with file-per-process access
- ❑ Stage 3: file-per-process read files from stage 2 and write #(procs/2) 32G files with shared access
- ~6TiB data for 16 nodes
- ~160 GiB/s read b/w
- ~130 GiB/s write b/w

MuMMI-like DAG I/O on Lassen's GPFS



- Dataflow with 4 stages
- Shared and file-per-process write in last stage
- Each file is 32G
- ~4TiB data for 16 nodes
- ~34 GiB/s read b/w for 16 nodes
- ~5 GiB/s write b/w for 16 nodes

Outline

- Understanding HPC Workflow I/O
- Wemul: HPC Workflow I/O Emulation Framework
- Experimental Results
- **Future Work**

Future Work

- Enable Wemul to generate workload in finer I/O pattern granularity
- Provide OpenMP support for multi-threading in DL training
- Enable staging and unstaging of checkpoint files using AXL
- Automatically generate the workflow definition through DAG
- Add support for other parallel I/O interfaces, i.e., HDF5, NetCDF, ADIOS, etc.
- Any additional suggestion of extensions helpful for the HPC community

Acknowledgements

- Thanks a lot for your time!

- Wemul source code is available in LLNL's GitHub
  - https://github.com/LLNL/Wemul

- Any questions, suggestions, feedback?
  - Create GitHub issue here: https://github.com/LLNL/Wemul/issues
  - Directly email to: fchowdhu@cs.fsu.edu

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-813999.