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-depending among workflow modules
Simple Workflow: Producer-Consumer I/O

- 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--input_dir &lt;path&gt;</td>
<td>Mountpoint or path to storage system to use</td>
</tr>
<tr>
<td>--block_size &lt;size in bytes&gt;</td>
<td>Block size per read or write request</td>
</tr>
<tr>
<td>--segment_count &lt;number&gt;</td>
<td>Total number of blocks or segments</td>
</tr>
<tr>
<td>--use_ior (optional)</td>
<td>Enable using IOR as a library</td>
</tr>
<tr>
<td>--num_epochs &lt;number&gt;</td>
<td>Number of epochs in DL training experiment</td>
</tr>
<tr>
<td>--comp_time_per_epoch &lt;time in seconds&gt;</td>
<td>Computation emulation per epoch</td>
</tr>
</tbody>
</table>
Wemul: Execution Modes (contd.)

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--inter_node</td>
<td>Set for enabling inter-node producer-consumer</td>
</tr>
<tr>
<td>--producer_only</td>
<td>Run Wemul as standalone producer application</td>
</tr>
<tr>
<td>--consumer_only</td>
<td>Run Wemul as standalone consumer application</td>
</tr>
<tr>
<td>--ranks_per_node &lt;number&gt;</td>
<td>Feed ranks per node number to help intra- or inter-node data transfer</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--read_input_dirs &lt;dir1:dir2::..&lt;&gt;</td>
<td>Colon separated list of mountpoints to storage systems for reading</td>
</tr>
<tr>
<td>--read_filenames &lt;file1:file2::..&lt;&gt;</td>
<td>Colon separated list of files to be read</td>
</tr>
<tr>
<td>--read_block_size &lt;size in bytes&gt;</td>
<td>Block size for the files to be read</td>
</tr>
<tr>
<td>--read_segment_count &lt;number&gt;</td>
<td>Segment count for the files to be read</td>
</tr>
<tr>
<td>--file_per_process_read</td>
<td>Enable file-per-process read (shared read by default)</td>
</tr>
<tr>
<td>--write_input_dirs &lt;dir1:dir2::..&lt;&gt;</td>
<td>Colon separated list of mountpoints to storage systems for writing</td>
</tr>
<tr>
<td>--write_filenames &lt;file1:file2::..&lt;&gt;</td>
<td>Colon separated list of files to be written</td>
</tr>
<tr>
<td>--write_block_size &lt;size in bytes&gt;</td>
<td>Block size for the files to be written</td>
</tr>
<tr>
<td>--write_segment_count &lt;number&gt;</td>
<td>Segment count for the files to be written</td>
</tr>
<tr>
<td>--file_per_process_write</td>
<td>Enable file-per-process write (shared write by default)</td>
</tr>
</tbody>
</table>
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>`
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](https://github.com/LLNL/Wemul)

- Any questions, suggestions, feedback?
  - Create GitHub issue here: [https://github.com/LLNL/Wemul/issues](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.

Disclaimer
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.