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

Fahim Chowdhury
Florida State University
fchowdhu@cs.fsu.edu

Francesco Di Natale
Lawrence Livermore Nat'l Lab
dinatale3@llnl.gov

Adam Moody
Lawrence Livermore Nat'l Lab
moody20@llnl.gov

Elsa Gonsiorowski
Lawrence Livermore Nat'l Lab
gonsiorowski1@llnl.gov

Kathryn Mohror
Lawrence Livermore Nat'l Lab
mohror1@llnl.gov

Weikuan Yu
Florida State University
yuw@cs.fsu.edu

1 INTRODUCTION

In general, workflow can be defined as a set of tasks, with or without dependencies on each other, to be executed in some predefined or random order. For instance, a complete execution of a set of applications run on a high performance computing (HPC) facility can be represented as a workflow. Oftentimes, one purpose can be served by multiple data dependent applications running via the same resource allocation. These scenarios can engender extremely complicated workflows and give rise to different types of data transfers or dataflows, e.g., inter-application, intra-application, inter-process, etc. Sometimes, these dataflows can become the bottleneck and hamper the overall performance of the application or set of interdependent applications.

An HPC cluster can be equipped with heterogeneous storage systems, consisting of on-node NVMe devices, burst buffers, parallel file systems (PFS), etc. To achieve efficient utilization of storage systems by HPC applications for facilitating dataflow, we need a complete understanding of the I/O behavior in application workflows. For example, I/O patterns with high numbers of metadata operations, e.g., as generated by deep learning (DL) applications, can result in high loads on PFSs. Again, in-situ and in-transit analysis of data in scientific application workflow can create I/O bottlenecks through producer-consumer like patterns. Systematic characterization and performance analysis can expose the intrinsic HPC I/O behavior of applications and inform I/O optimization decisions. The goal of this project is to extract the I/O characteristics of various HPC workflows and develop strategies to optimize overall application performance by improving the I/O behavior.

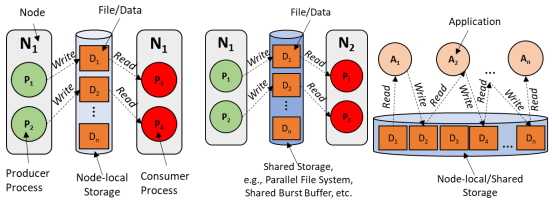
To precisely understand the I/O behavior in HPC workflows, in this work, we make the following efforts:

- We develop a dataflow emulator to generate different types of HPC workloads.
- We characterize the I/O patterns posed by different workloads generated by the emulator.
- We analyze the Cancer Moonshot Pilot 2 (CMP2) project's workflow and I/O demands.

2 EMULATING HPC I/O PATTERNS

Deep Learning Training I/O. In the training phase of DL applications' workflow, the dataset import pipeline needs to read all or portion of the files in the dataset at the beginning of every epoch. Usually, these data files are tiny (i.e. 100KB) [**?**] and persistently kept in the PFS. Hence, it creates a lot of metadata overhead compared to the data read. Each reader process is assigned an exclusive list of files to read from the PFS. Again, for avoiding bias in the stochastic gradient descent step of the training, it is mandatory to randomly shuffle all the files at the starting of each epoch.

Producer-Consumer I/O. This type of I/O pattern is available in workflows with in-situ or in-transit analysis of data. [**?**] From system's perspective, some processes in a node of HPC cluster are assigned to write data and some other processes on the same node are responsible for reading those data for further analysis as shown in Figure **??**. On the other hand, in some cases, the producer and consumer processes can reside in different nodes, i.e., Figure **??**, and the dataflow has to go through network communication. As shown in Figure **??**, from the application's perspective, the applications in a workflow can be connected by a chain of I/O requests, where one component of the workflow writes some data and the next component starts execution by taking those data as input.



(a) Node-local (b) Inter-node (c) I/O chain

Figure 1: Producer-Consumer I/O pattern

Checkpoint/Restart I/O. In case of checkpoint/restart I/O pattern, each process or a set of selected processes create checkpoints in a pre-defined frequency usually set by the application developer or user. In modern clusters, checkpoint files can be written to on-node fast persistent storage devices or staged via shared burst buffer file systems. Later, these staged data are flushed to PFS asynchronously. If any process crashes, the processes restart from the latest checkpointed state.

Dataflow Emulator. Currently, we are developing an MPI enabled C++ application to emulate DL, producer-consumer and checkpoint/restart I/O patterns for in-depth analysis. Interleaved-Or-Random (IOR) [**?**], an established I/O benchmarking tool is used as a static library from emulator for basic functionalities. The `dataflow_emulator`, derived from generic `workflow_emulator` is the factory for creating different types of I/O workloads according to user input. On the other hand, `ior_runner` class is responsible

for calling the IOR library functions. Asynchronous Transfer Library (AXL) [?] is leveraged as a library from `checkpoint_restart` module for transferring files while stage-in and out phase. This application is currently under active development.

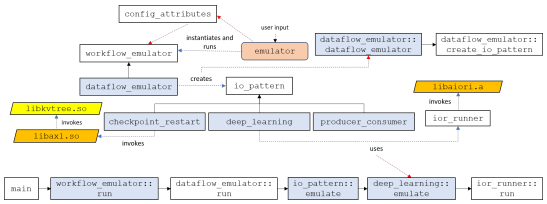


Figure 2: Class Diagram of Dataflow Emulator

3 DATAFLOW IN CANCER MOONSHOT PILOT 2

The target of cancer moonshot project is the advancement of cancer diagnosis by leveraging HPC systems. [?] In particular, the CMP2 project is being developed to simulate the RAS protein and cell membrane interaction for assisting early stage cancer cell detection. This workflow is managed by MaestroWF [?] workflow manager along with Flux resource manager [?]. It consists of three steps, where it first reads in data from a macro analysis to construct patch files. These files are treated as input to the further coarse-grained (CG) particle analysis. After the CG setup, format of the input patches are converted and passed to the CG simulation step. Thus, the dataflow in the CMP2 workflow creates an I/O chain as shown in Figure **??**.

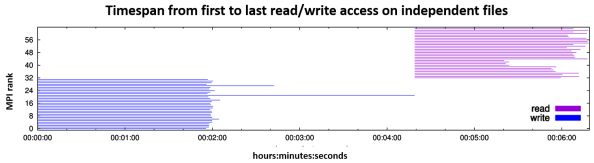
4 EXPERIMENTAL RESULTS

To understand and visualize different types of I/O patterns generated by the emulator, we run emulator on Lassen [?] with Darshan [?] profiling tool. For all the tests, we increase the number of nodes from 1 to 16 with 4 processes per node.

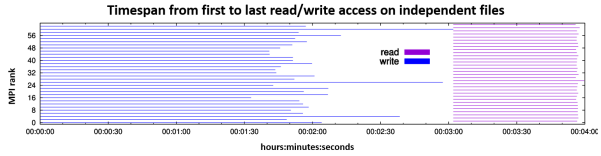
In Figure **??**, we show the time line of read operations by each process involved in the 16 nodes emulator run for reading a sample dataset in the General Parallel File System (GPFS) deployment on Lassen. The dataset has 320 dummy files of 1 MiB size each arranged in a single-depth hierarchical directory structure. We can visualize the tiny file read requests that are performed by each process to read the files assigned to it.

Figure 3: Deep Learning training I/O timeline

In Figure **??**, we observe the I/O access pattern in a producer-consumer type data dependency system where a group of processes write one file per process and another group reads those files afterward. Figure **??** shows that, there is a time gap between write and read operations which hampers the final execution time if the producer and consumer processes reside on different nodes, and files are transferred through PFS. On the other hand, Figure **??** demonstrates better execution time when the producer and consumer processes are co-located on the same node. Again, according to Figure **??**, the runtime is improved even more when burst buffer is used to transfer files.



(a) Inter-node PFS



(b) Intra-node PFS

(c) Intra-node Burst Buffer

Figure 4: Producer-Consumer I/O timeline

The checkpoint/restart I/O pattern can be visualized by Figure **??** where one process writes checkpoint files on a directory of the GPFS mountpoint and randomly stops to emulate a system crash. Afterward, all the processes emulate a restart and searches for the last modified checkpoint file and read its contents.

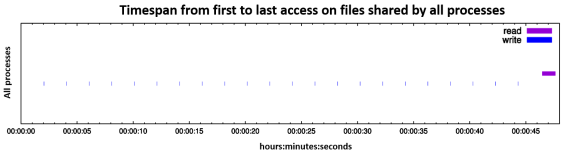


Figure 5: Checkpoint/Restart I/O timeline

5 CONCLUSION AND FUTURE WORKS

Scientific application workflows on HPC systems can be difficult to understand. Dataflow caused by HPC I/O via the workflows can be equally complicated to handle. Effective I/O management system in a workflow manager like MaestroWF can possibly improve the overall application performance. Third party API libraries like AXL [?] can also be modified for intelligent data transfer among different units of heterogeneous storage resources in leadership supercomputers. In future, we plan to profile CMP2 workflow using tools like Darshan [?] or Recorder [?] to detect the I/O behavior and dataflow. Later, we will develop data management strategies to properly handle the dataflow in complicated and composite HPC workflows like CMP2.