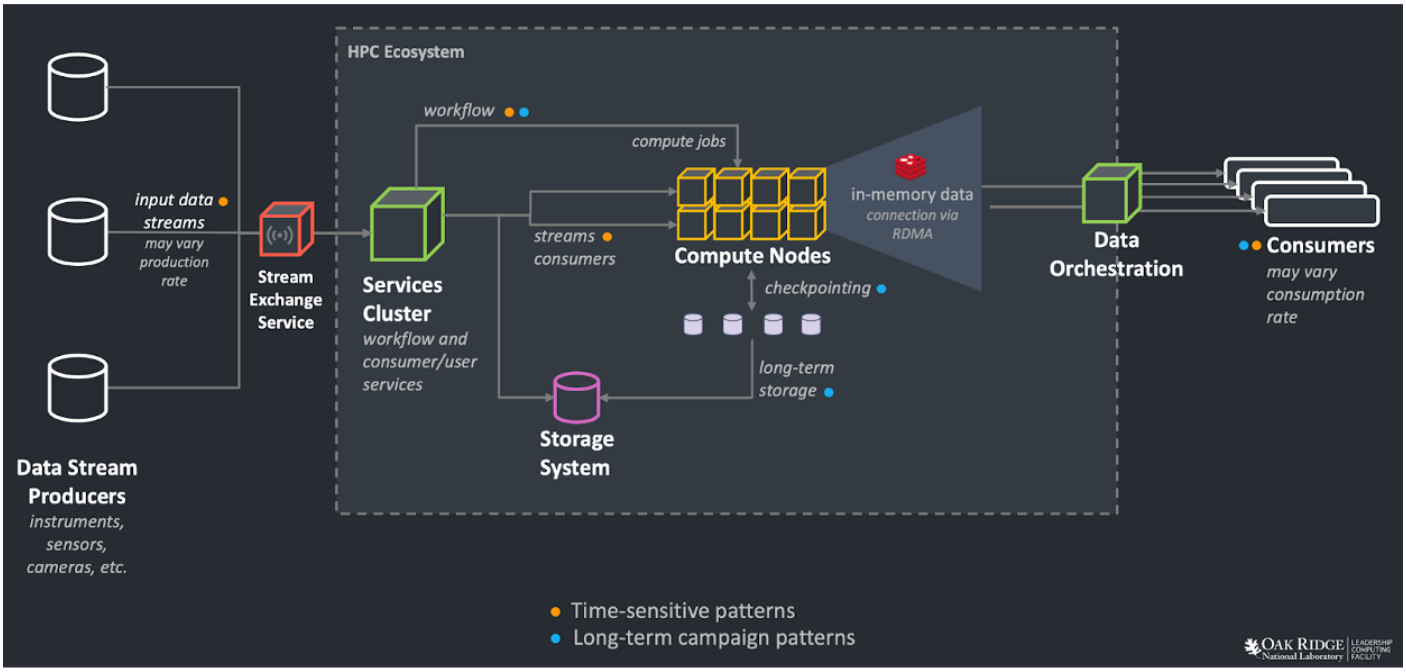
OLCF-6 Workflow Benchmark

- Patrick Widener (widenerpm@ornl.gov (mailto:widenerpm@ornl.gov))
- Junqi Yin (yinj@ornl.gov (mailto:yinj@ornl.gov))
- Ketan Maheshwari (maheshwarikc@ornl.gov (mailto:maheshwarikc@ornl.gov))
- Rafael Ferreira da Silva (silvarf@ornl.gov (mailto:silvarf@ornl.gov))

Purpose of Benchmark

The objective of this workflow benchmark is to assess the capability of the High-Performance Computing (HPC) system in supporting dynamic workloads that originate from various data stream sources. These workloads will be processed within the compute nodes. The processed data will then be made available to a wide array of consumers, each potentially consuming unique abstractions of the data. We target an ML4NSE (Machine Learning for Neutron Scattering Experiment) application that employs a Temporal Fusion Transformer (TFT) model to both train on and predict the measurement time for a distinct cluster of peaks. This cluster includes a robust nuclear peak along with six weaker satellite peaks, resulting from the magnetic ordering within a single-crystal sample. The primary objective of this code is to enable near real-time decision-making by leveraging the combined powers of Machine Learning (ML) and High-Performance Computing (HPC). This benchmark provides a self-contained, end-to-end evaluation of a coupled compute/data problem.



Abstract overview of a reference implementation of the workflow benchmark.

Characteristics of Benchmark

Data is channeled from multiple sources to a gateway node using an open-source streaming transport layer capable of handling structured data, not just byte sequences (e.g., ZeroMQ, RabbitMQ). One input stream to the gateway node may potentially serve as a control stream, directing how to multiplex, filter, window, or otherwise service the other data streams. This orchestrates the creation of a single stream of structured data, possibly distinct in structure from the input streams, which is then forwarded to a listener at the Services Cluster, ensuring efficient information flow.

The integration between the source streams (at the edge/service nodes) and the ML4NSE (running within the compute nodes) facilitates the swift analysis of data generated during an experiment, offering timely feedback to researchers (consumers). Such prompt insights allow for the precise adjustment of experimental parameters, enhancing the effectiveness and responsiveness of the ongoing research.

The Data Orchestrator process has connectivity to both the fast, reliable internal network of the Compute Cluster (e.g. RDMA but potentially other technology) and a network physically disjoint from that internal network. It can potentially communicate with any process in the job running on the Compute Cluster. Periodically the Data Orchestrator will create output data events based on the inputs it receives from the compute job. These events are forwarded to the current subscriber set on one or more output data streams. The output events may be customized (filtered/windowed/transformed) differently for different data streams (for instance, a particular output stream may have a guarantee of no more than one event per wall-clock second, while another output stream may only send summarized data based on input from all processes in the compute job).

Mechanics of Building Benchmark

```
bash setup.sh
source env.sh
pip install -r requirements-torch.txt
pip install -r requirements.txt
```

Software Dependencies

Dependencies for the benchmark are listed in the following files:

`env.sh`
`requirements-torch.txt`
`requirements.txt`
Specifically, the benchmark depends on:

- GCC 10.3+
- ROCm 6.0.0
- Python 3.8+
  - torch 2.3.1+rocm6.0
  - torchvision 0.18.1+rocm6.0
  - torchaudio 2.3.1+rocm6.0
  - pytorch_lightning 2.3.0
  - pytorch_forecasting
  - numpy 1.26.4
  - pandas 1.5.3
  - pyyaml
  - pyzmq
  - matplotlib
  - scikit-learn
  - optuna_integration

Mechanics of Running Benchmark

The benchmark tests were consistently performed utilizing the Frontier supercomputer at Oak Ridge National Laboratory (ORNL).

To initiate the benchmark execution, the primary step involves activating the data stream service. It is important to note that, for the purpose of testing, the data stream service has been installed on the login node of the Frontier system. The activation of this service can be achieved through the following command (preferably to be run on a separate shell):

```
source env.sh
python exchanger.py
```

The `job.sb` job description file provides the necessary steps for requesting computing nodes and setting the environment to run the benchmark.

```
sbatch job.sb
```

Once the job starts running, it will hold waiting for the data stream. To start streaming the data, execute the following command:

```
python sender.py
```

Changing the Scale of the Benchmark

The benchmark's scale can be adjusted by altering the number of replicas in the workflow execution. At its smallest scale, the benchmark utilizes 9 nodes (a single replica). To increase the number of replicas, modify the `job.sb` submission script by updating the node count and the number of replicas. For example, to use 900 nodes and 100 replicas, adjust the script to `#SBATCH -N 900` and set `REPLICAS=100` accordingly.

Run Rules

- *Weak Scaling Experiments:* Each rank at level 1 (refer to Figure 2) trains a TFT model on 64 ([4, 4, 4]) voxels, and the level 2 rank operates on [2, 2, 2] mean voxels of level 1. Consequently, for an input with dimensions 8 × 8 × 8, a total of 9 ranks (eight in level 1 and one in level 2) are required. For a larger input of 64 × 64 × 64, a total of 4608 ranks are needed, divided into 4096 in level 1 and 512 in level 2.

Figure of Merit

The primary figure of merit for the ML4NSE (defined in detail in https://doi.org/10.1615/JMachLearnModelComput.2023048607 (https://doi.org/10.1615/JMachLearnModelComput.2023048607)) workflow benchmark is the number of voxels per second: `(#voxels * #replicas) / (workflow_makespan)` (workflow makespan represents the total time to run all replicas in the workflow.)

Some useful secondary figures of merit will be:

- Exchange bandwidth (ingress) at the gateway node which multiplexes among input streams

- Scalability of input/output stream multiplexing – ideally, data rates at the gateway nodes on both sides of the application workflow will degrade linearly with the number of input/output streams.

An additional potential figure of merit that could demonstrate the robustness of the system would include any active guidance between the Compute and Services Clusters; the latency involved in control operations becomes crucial. Specifically, it's essential to assess the duration a compute job is held while disseminating new control information. Also, as additional input streams and output consumers are added, the effect on end-to-end time-to-solution could be affected.

Dataset	Dimension	#Nodes	#Replicas	Send. Transfer Rate	Avg. Rec. Transfer Rate (per rank)	FOM (#voxels*#replicas/second)
p_322_data_np_res_16.npy	8 x 8 x 8	9	1	61.14 Gbps	133.80 Mbps	0.51
p_322_data_np_res_16.npy	8 x 8 x 8	90	10	60.01 Gbps	5.57 Mbps	4.74
p_322_data_np_res_16.npy	8 x 8 x 8	900	100	59.94 Gbps	2.91 Mbps	40.63