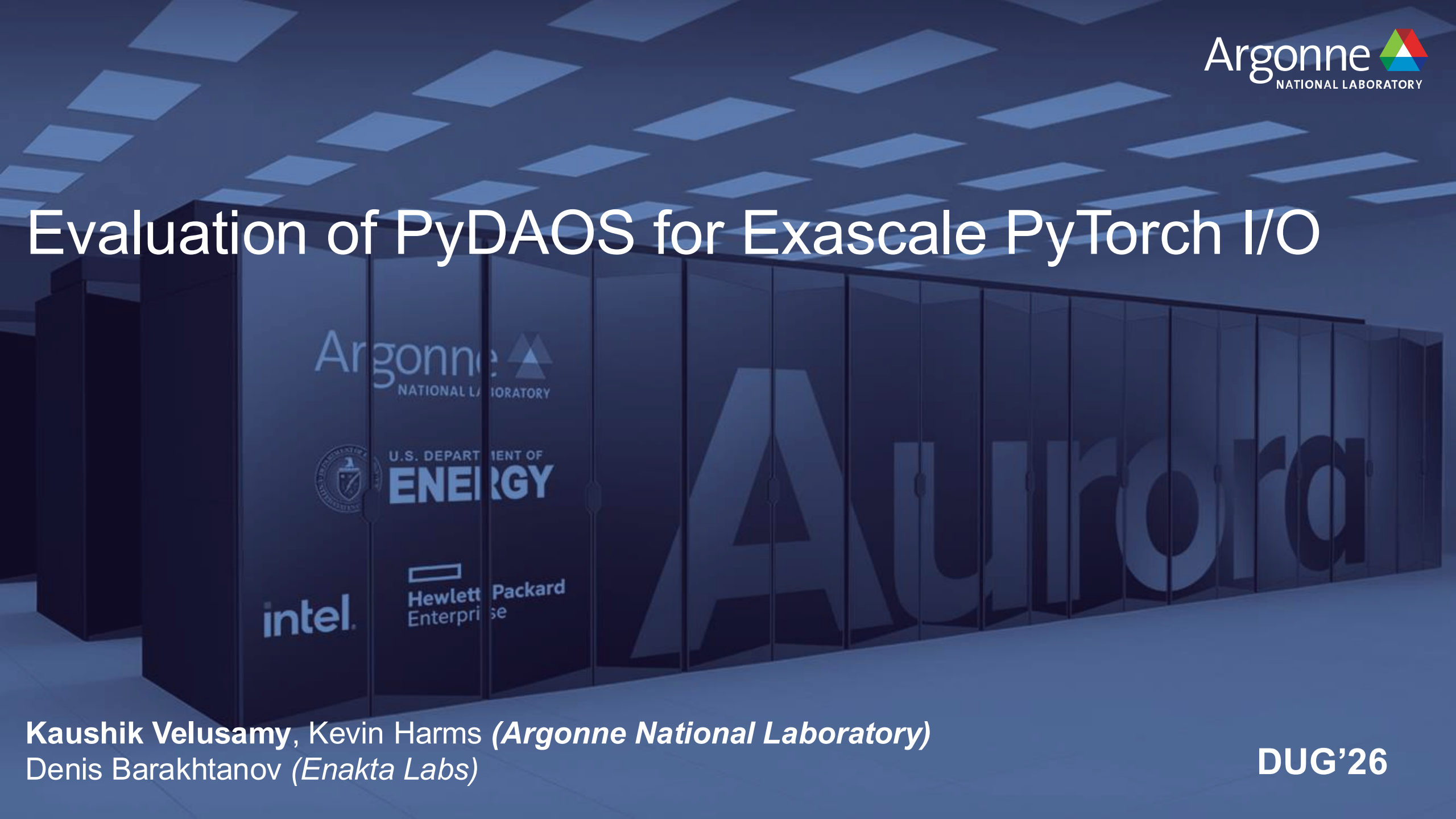


Evaluation of PyDAOS for Exascale PyTorch I/O

Kaushik Velusamy, Kevin Harms *(Argonne National Laboratory)*
Denis Barakhtanov *(Enakta Labs)*

DUG'26

I/O Modes for PyTorch

DATASET LOADING
(READ)

MODEL LOADING
(READ)

CHECKPOINT
(WRITE)

Argonne NATIONAL LABORATORY

Standard PyTorch style with a simple NPZ Dataset class

- Example of standard Dataset used with DataLoader
- Can use directly with dfuse and POSIX intercept library
- Note Dataset encodes:
  - I/O model
  - Data model
  - Data format

```python
import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader

class NpzDataset(Dataset):
    def __init__(self, root):
        self.root = root
        self.files = sorted(
            os.path.join(root, f) for f in os.listdir(root) if f.endswith(".npz")
        )

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        arr = np.load(self.files[idx], allow_pickle=True)["x"]
        return torch.tensor(arr)

# regular DataLoader
ds = NpzDataset("/path/or/mounted/daos/dataset")
loader = DataLoader(ds, batch_size=32, shuffle=True, num_workers=4)

# regular checkpoint save/load
state = {"model": torch.ones(8)}
torch.save(state, "/path/or/mounted/daos/checkpoints/ckpt.pt")
loaded = torch.load("/path/or/mounted/daos/checkpoints/ckpt.pt", weights_only=True)
```


Argonne NATIONAL LABORATORY

pydaos.torch PyTorch Dataset

- PyTorch defines a Dataset abstraction which encapsulates all I/O for data
- Datasets are passed to a the DataLoader which coordinates loading of data
- The Dataset has essentially one abstraction, __getitem[s]__
  - I/O performed in this call
- Transform function allows user to provide “business logic” to interpret the data read from the system
- pydaos.torch.Dataset()

- A shim layer is implemented that translates between python and DFS API calls

```
dataset = DaosDataset(pool=pool, cont=cont, transform_fn=transform)

loader = DataLoader( dataset, batch_size=batch_size,
                              shuffle=shuffle, num_workers=num_workers,
                              pin_memory=torch.cuda.is_available(), )
```

https://github.com/daos-stack/daos/tree/master/src/client/pydaos/torch

Argonne NATIONAL LABORATORY

PyDAOS - PyTorch

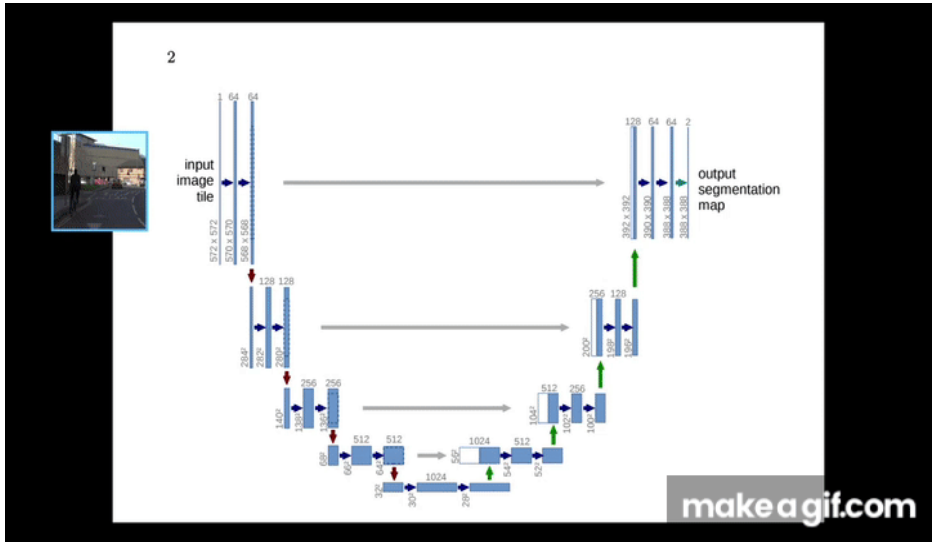
- The dataset is opened by DAOS pool/container name
- Any dataset must be converted into a “DaosDataset”
  - Business logic encoded in transform function
- pydaos.torch.Dataset will support reading all files in a directory
- PyDAOS dataset uses DFS directly and bypasses all POSIX calls
- Checkpoint works directly as would by another checkpoint writer

```python
import numpy as np
import torch
from io import BytesIO
from pydaos.daos_torch import Dataset as DaosDataset
from pydaos.daos_torch import Checkpoint as DaosCheckpoint

def transform(data: bytes):
    return np.load(BytesIO(data), allow_pickle=True)["x"]

# DAOS-native-ish dataset handle
ds = DaosDataset(pool="datascience", cont="my-dataset", transform_fn=transform)

# checkpoint handle bound to DAOS pool/container
ckpt = DaosCheckpoint("datascience", "my_container", "")

state = {"model": torch.ones(8)}
name = "/data-0-of-1.pt"

with ckpt.writer(name) as f:
    torch.save(state, f)

stream = ckpt.reader(name)
loaded = torch.load(stream, weights_only=True)
```


Argonne NATIONAL LABORATORY

Unet3D – Example

- a **U-Net 3D** is an artificial intelligence model designed to look at 3D images (like a stack of photos or a video) and identify exactly where specific objects are located.
- **A 3D U-Net is a specific type of CNN architecture.**
- **Goal:** To figure out exactly *where* the object is pixel-by-pixel (or "voxel-by-voxel" in 3D).
- Instead of a sliding window moving over height and width the 3D U-Net uses a window that slides through height, width, and depth.
- These layers are responsible for detecting features like edges, textures, and shapes across volumetric data (like an MRI scan).



Example Use Cases:

- Medical Imaging (MRI & CT Scans) separating healthy tissue from a tumor.
- Self-Driving Cars (LiDAR)
- Biology (Microscopy) 3D scans of cells.

Argonne NATIONAL LABORATORY

DLIO (Deep Learning I/O) Benchmark

- From the DLIO README
  - *DLIO is aimed at emulating the I/O behavior of various deep learning applications. The benchmark is delivered as an executable that can be configured for various I/O patterns. It uses a modular design to incorporate more data loaders, data formats, datasets, and configuration parameters. It emulates modern deep learning applications using Benchmark Runner, Data Generator, Format Handler, and I/O Profiler modules.*
- DLIO is used to exercise loading the Unet3D example dataset via the PyTorch framework
- DLIO does not execute any training on the GPU
  - Instead, it sleeps to emulate the training cycle
- DLIO not designed to generate peak I/O bandwidth, but execute the characteristic I/O load that is seen in Deep Learning applications
- https://github.com/argonne-lcf/dlio_benchmark

Argonne NATIONAL LABORATORY

Parameters and Configuration

DLIO Configuration

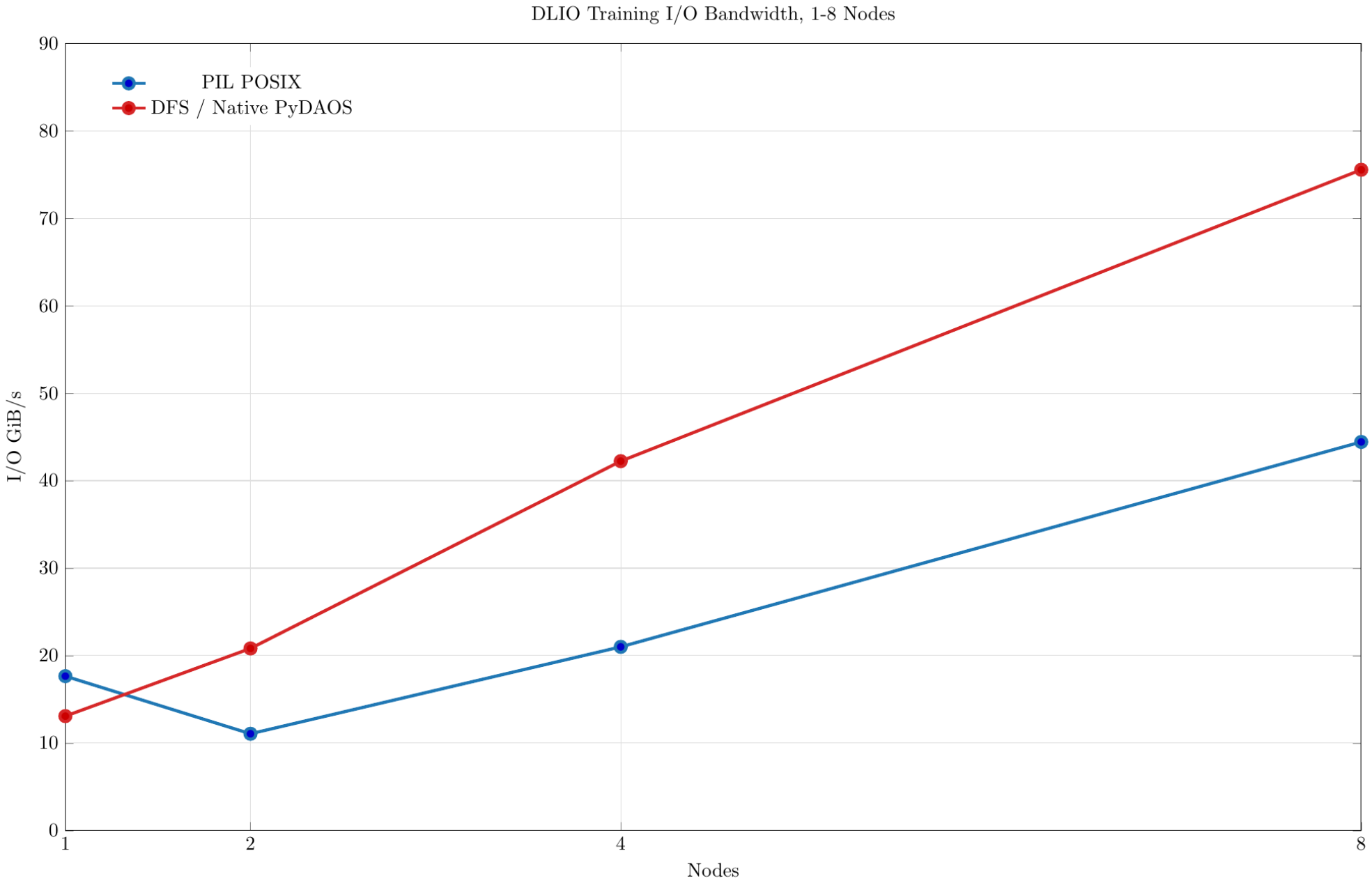
- batch_size = 7
- GPU_per_node = 12
- samples/file = 1
- steps = 100
- file_size = 146600628 bytes
- sleep_time = 0.093254 s

Per Epoch

- 7 * 12 * samples/file * file_size * nodes
- For 8 nodes
  - = 7 * 12 * 1 * 139.8 MiB * 8
  - = 91.7 GiB

Nodes	GPUs / NP	Global Batch	Steps	Num Files Train	Total Samples	Approx Total Size
1	12	84	100	8,400	8,400	1.12 TiB
2	24	168	100	16,800	16,800	2.24 TiB
4	48	336	100	33,600	33,600	4.48 TiB
8	96	672	100	67,200	67,200	8.96 TiB
16	192	1,344	100	134,400	134,400	17.92 TiB
32	384	2,688	100	268,800	268,800	35.84 TiB
64	768	5,376	100	537,600	537,600	71.68 TiB
128	1,536	10,752	100	1,075,200	1,075,200	143.36 TiB
256	3,072	21,504	100	2,150,400	2,150,400	286.72 TiB
512	6,144	43,008	100	4,300,800	4,300,800	573.44 TiB

Argonne NATIONAL LABORATORY



Argonne NATIONAL LABORATORY

Acknowledgements

HPE DAOS Team.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

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.