DYAD (DYnamic and Asynchronous Data-streamliner)

SCA/HPCasia ‘26

01/26/2026

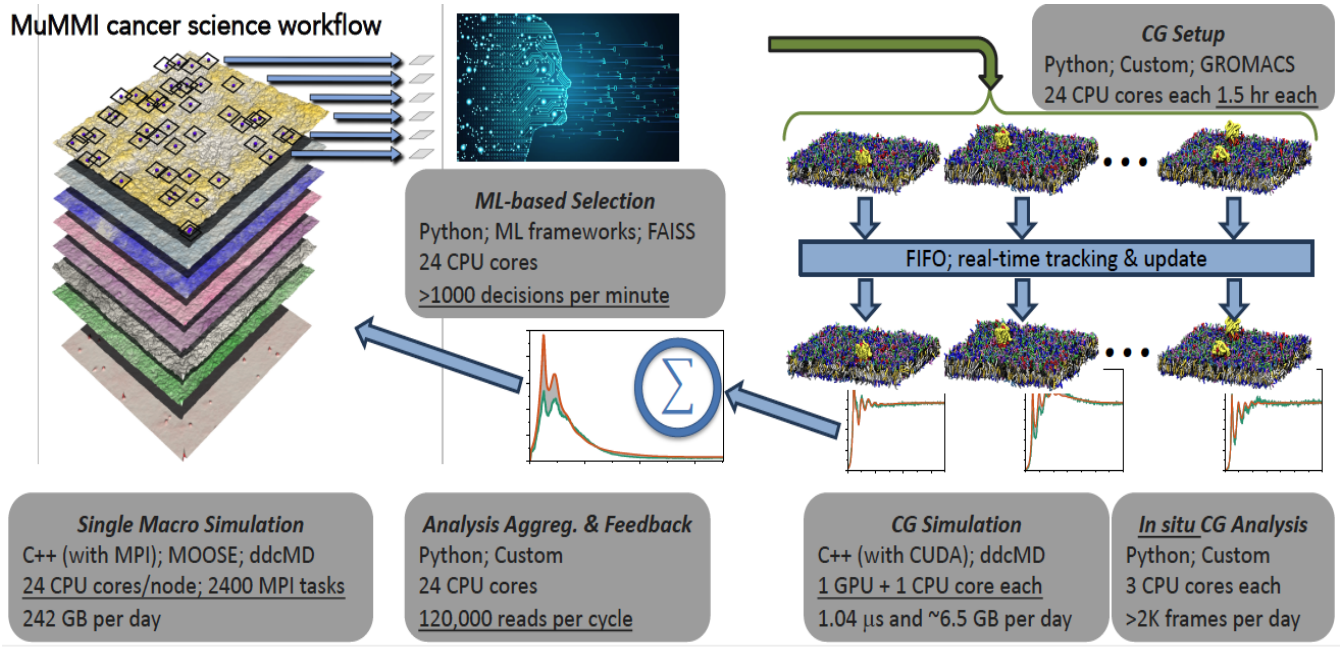
Jae-Seung Yeom
Computer Scientist
Center for Applied Scientific Computing/LLNL

LLNL-PRES-2015067
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Scientific workflows and AI

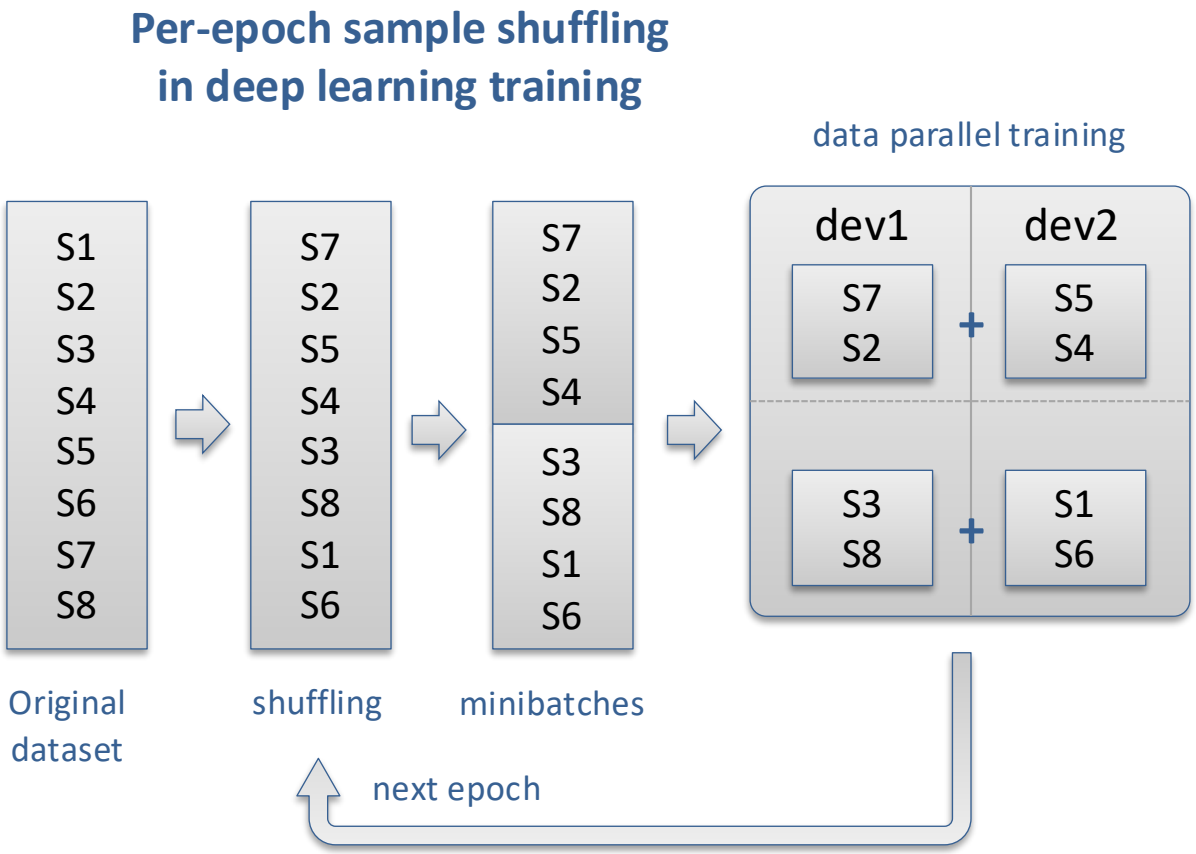
are emerging as the new dominant HPC workloads



- Shifting from monolithic large-scale MPI jobs to workflows
  - Ensembles of tasks
  - Some tasks depends on the completion and results of other tasks
- AI-driven steering of simulations
  - Sampling
  - Surrogate models
  - Analytics

Producer–consumer relationships between tasks in a workflow introduce data dependencies.
In practice, these data-sharing requirements are often satisfied using shared file systems.

Unique characteristics of I/O in deep learning training

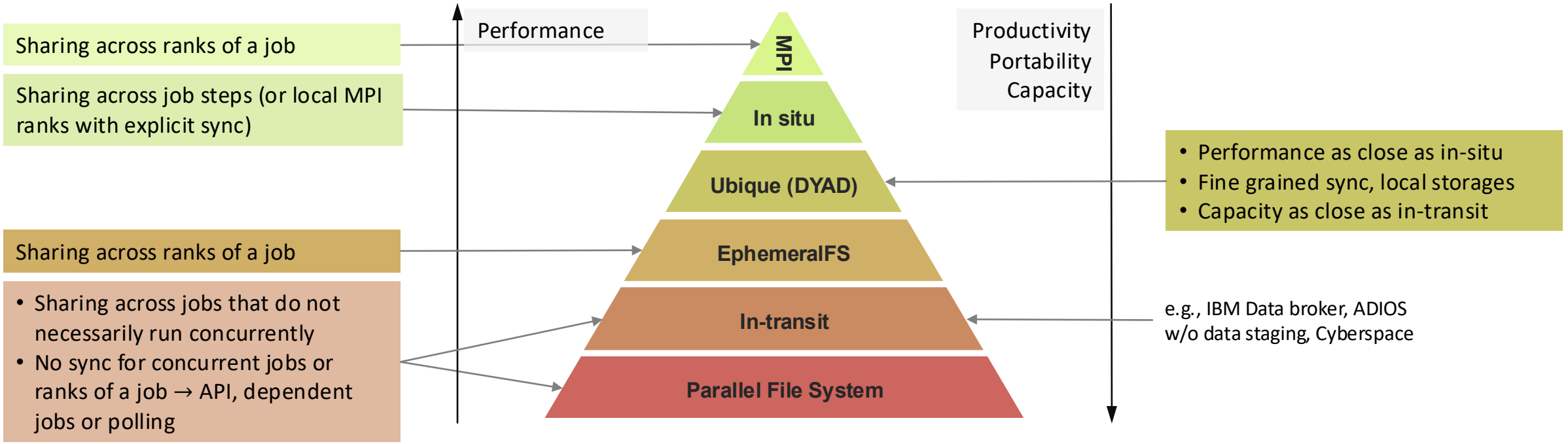


- Dominated by reads
- Large number of small files
- Reused data across epochs, and hyperparameter searches
- Random sample reads
- Concurrent and async reads
- Latency sensitive reads
- Data Parallel training shards data across multiple devices

In distributed training, data files are reused across epochs and randomly accessed by multiple devices, resulting in repeated reads.

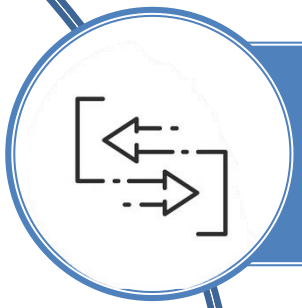
Design trade-off space for data transfer model

DYAD is an embodiment of Ubique model based on FLUX API



Shared storage provides decoupling, productivity and portability across tasks and nodes
DYAD balances the productivity, portability, and performance

DYAD: DL-Centric Solution (3 Goals)



Data Movement

- Maximizes network throughput with pinned buffered, lightweight control plane and RDMA-capable data plane.

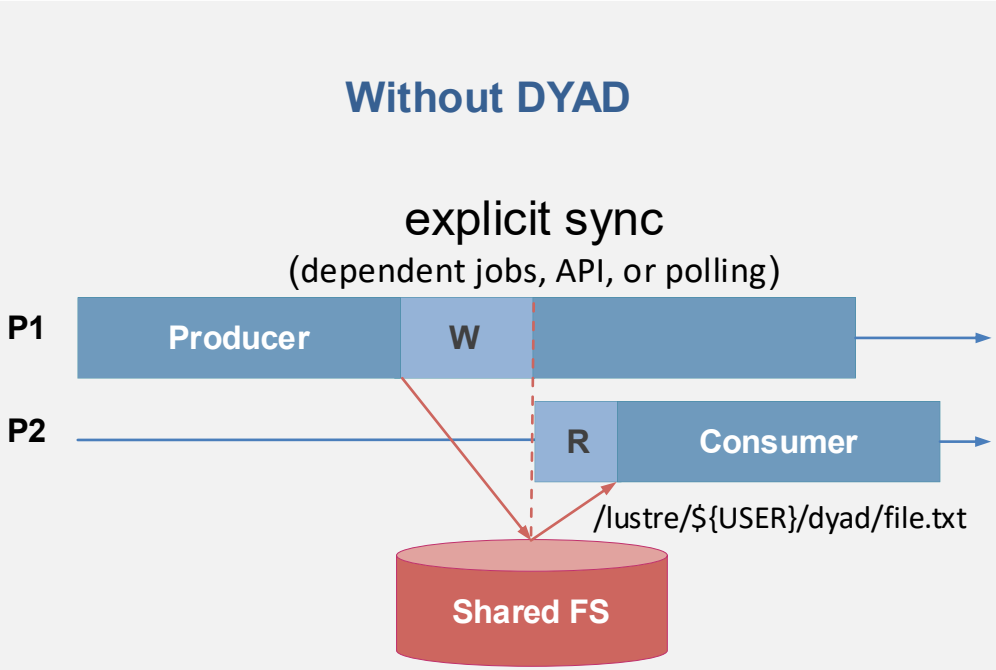
Cache Management (Increases Node-Local Accesses)

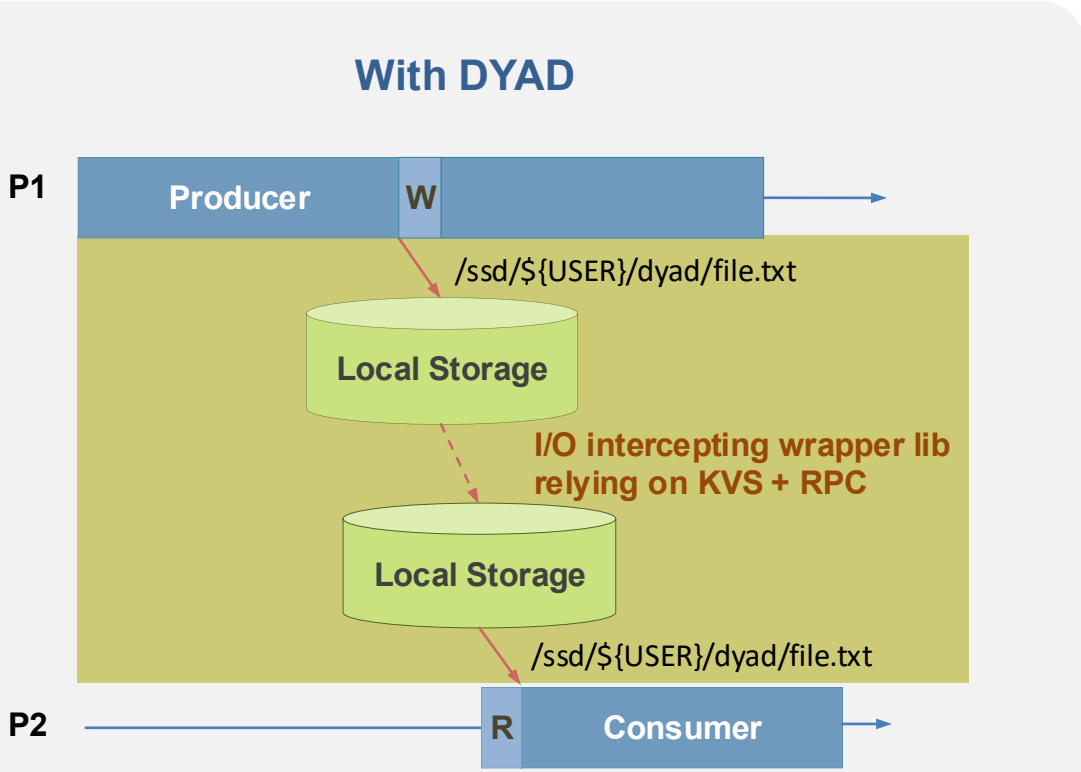
- Inclusive cache design
- Cache duplication for higher availability

Hierarchical Sample Discovery

- Isolating metadata queries
- Caching metadata accesses

Transparently leverage local storages to improve I/O performance

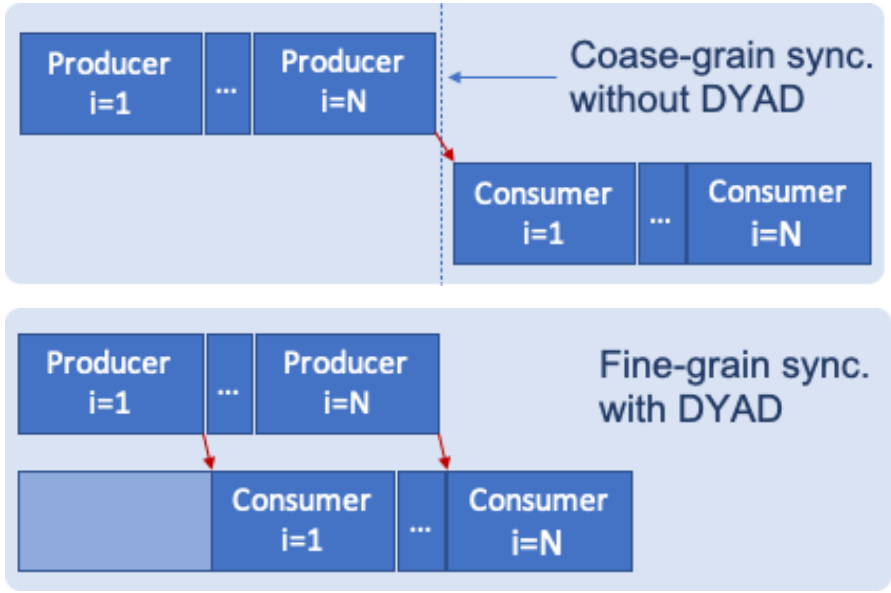




Fine-grain synchronization to improve data locality

```
producer() {
    for (i=1; i<=N; i++) {
        produce(data[i])
        write(data[i], file[i]);
    }
}
```

```
consumer() {
    for (i=1; i<=N;i++) {
        read(file[i], data[i]);
        consume(data[i])
    }
}
```



- Explore the effectiveness of the scheduling to minimize the **temporal** and the **spatial** distance between producers and consumers

Minimal example

- Running DYAD service with FLUX

  flux exec -r all flux module load dyad.so /ssd/${USER}/dyad

```
flux exec -r all flux module load dyad.so /ssd/${USER}/dyad
```

- Configure DYAD via env variables

```
DYAD_KVS_NAMESPACE
DYAD_PATH_PRODUCER
DYAD_PATH_CONSUMER
DYAD_SHARED_STORAGE
```

- Running user app written in C with DYAD

```
LD_PRELOAD=libdyad_sync.so:${LD_PRELOAD} ./app
```

- Using DYAD stream library in C++

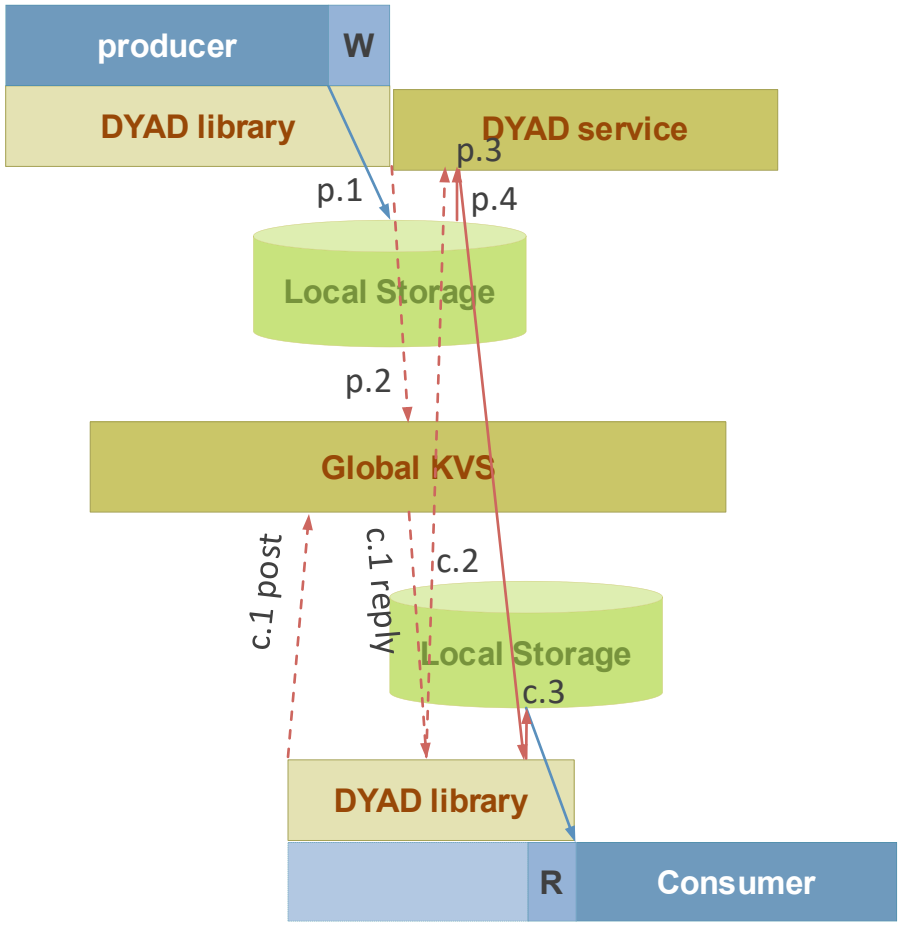
```
#include <dyad_stream_api.hpp>

void producer(dyad::dyad_stream_core& dyad)
{
   dyad::ofstream_dyad ofs_dyad;
   ofs_dyad.init(dyad);
   ofs_dyad.open("test.txt");
   std::ofstream& ofs = ofs_dyad.get_stream();
   ofs << "test" << std::endl;
   ofs_dyad.close();
}

void consumer(dyad::dyad_stream_core& dyad)
{
   dyad::ifstream_dyad ifs_dyad;
   ifs_dyad.init(dyad);
   ifs_dyad.open("test.txt");
   std::ifstream& ifs = ifs_dyad.get_stream();
   std::string line;
   ifs >> line;
   std::cout << line << std::endl;
   ifs_dyad.close();
}
```

```
std::ofstream ofs;
ofs.open("test.txt");
```

High-level description of operation



DYAD components

- DYAD service runs on each node.
- DYAD library only intercepts I/O for files on the DYAD Managed Directory (DMD)

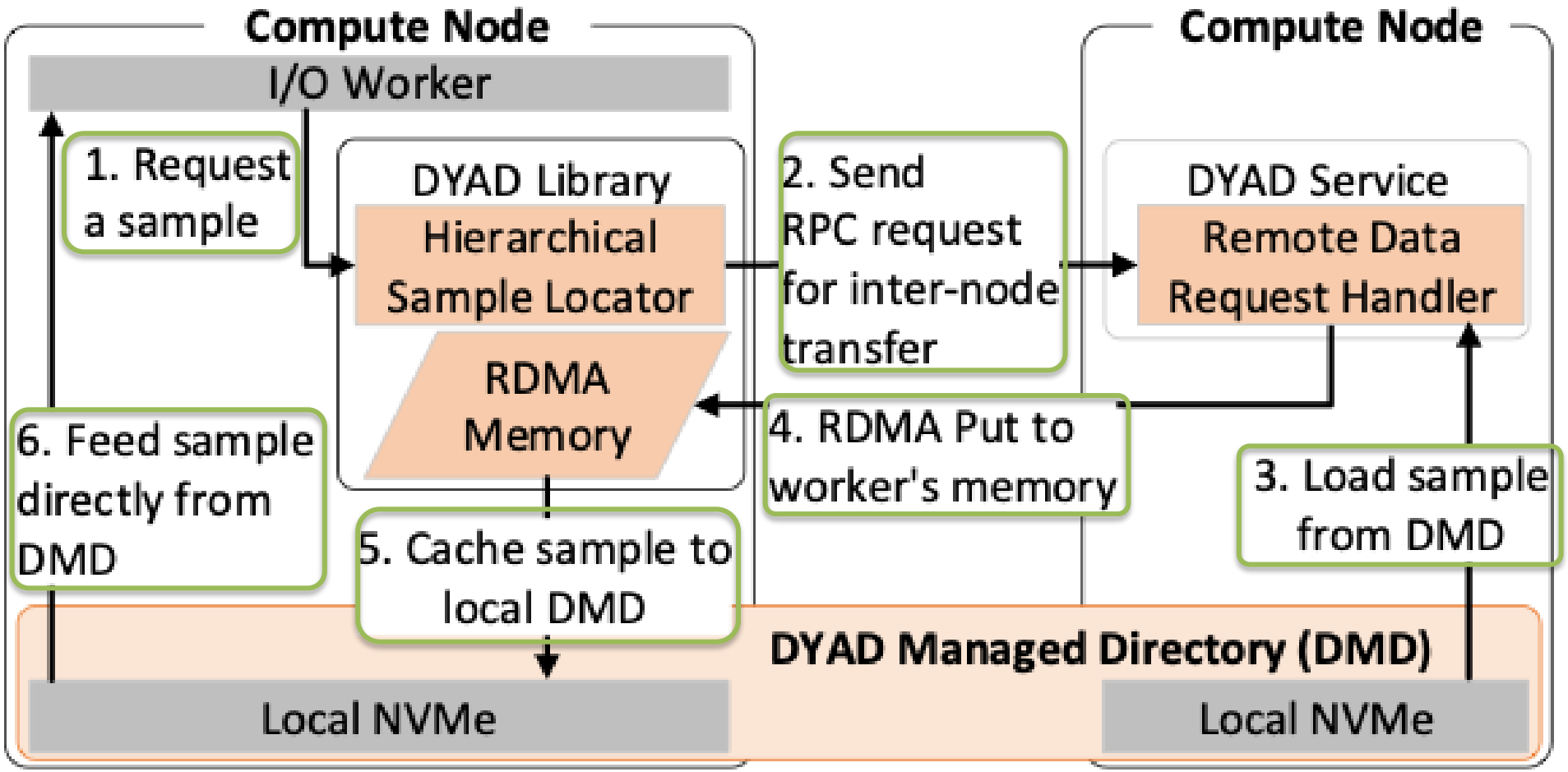
Library (producer side)

- p.1 *write(manged_dir/filenpath)*
- p.2 *publish(<filepath, prod_rank>)*
  - If a file is written into ***managed_dir*** or its subdirectory, DYAD inserts an entry into the global key-value-store (KVS)
  - The KVS entry is a pair of ***filepath*** and the owner ***rank***

Library (consumer side)

- c.1 *query(filepath)→ prod_rank*
  - client queries KVS to obtain the rank of the file owner (producer). Then, blocking wait.
- c.2 *rpc_get(prod_rank, filepath)*
  - Asks the owner to transfer the file.
  - p.3: owner service loads the requested file into buffer
  - p.4: service sends it to the client
- c.3 *store(managed_dir/filepath)*
  - Upon receiving, stores it on its LS and let the app resume normal read

High-Level Inter-node Data Movement for reading a remote file

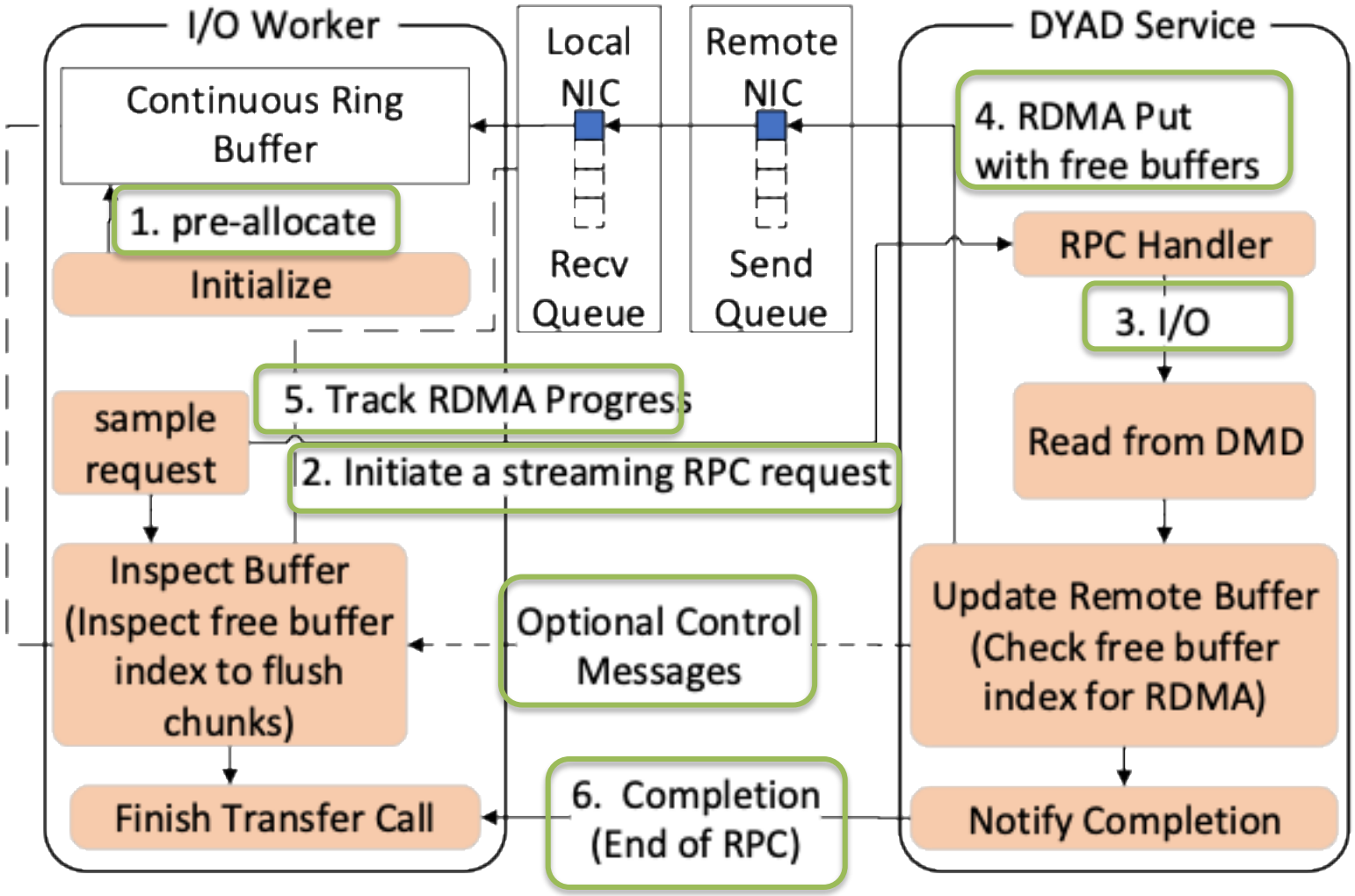


Client lib on consumer side

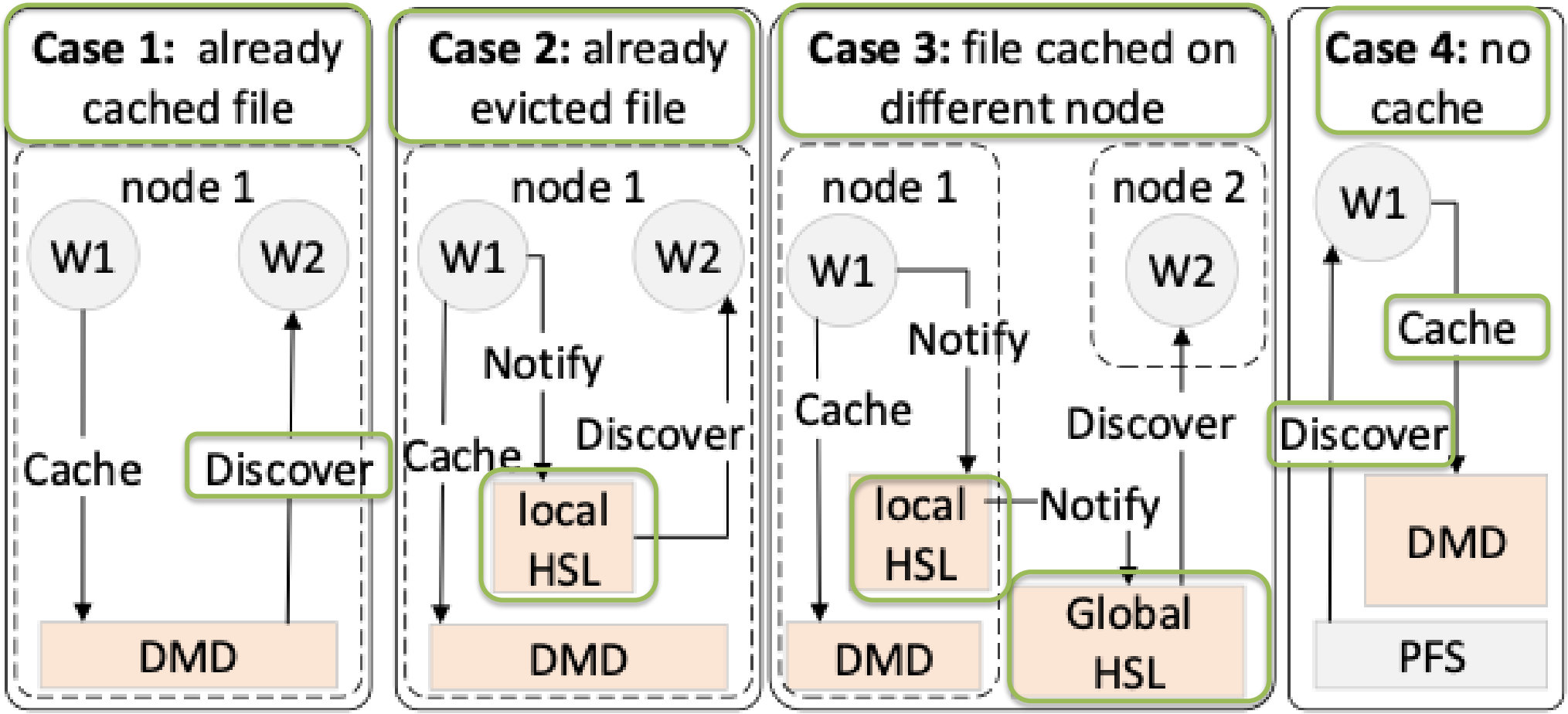
Service on owner side

Streaming RPC over RDMA protocol

- Avoid Dynamic Buffers
- Cache RDMA pin connections
- Avoid multiple control plane messages
- Utilize worker engine to track progress

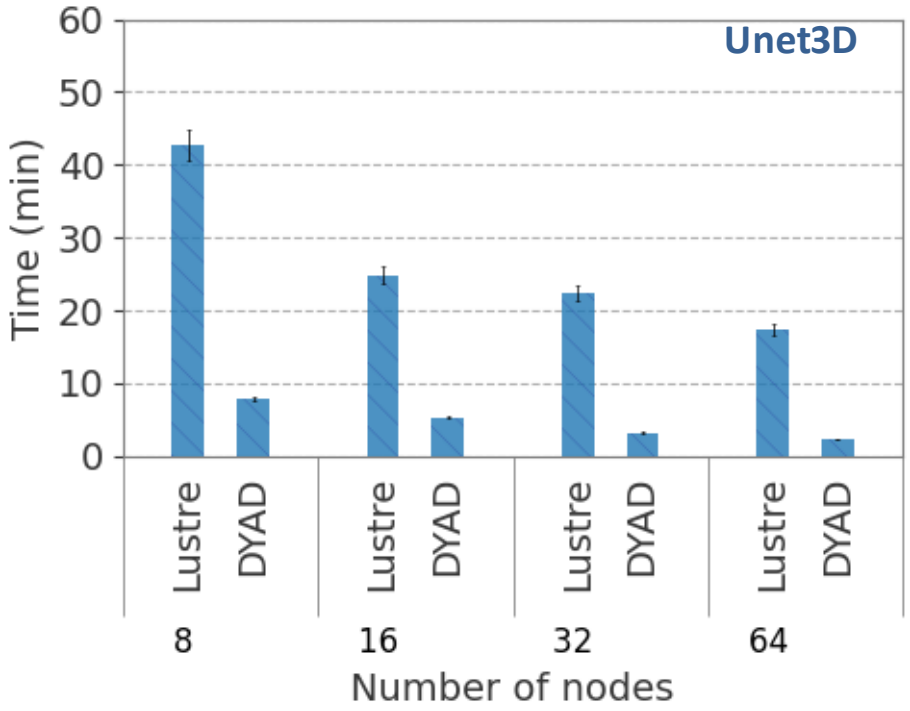


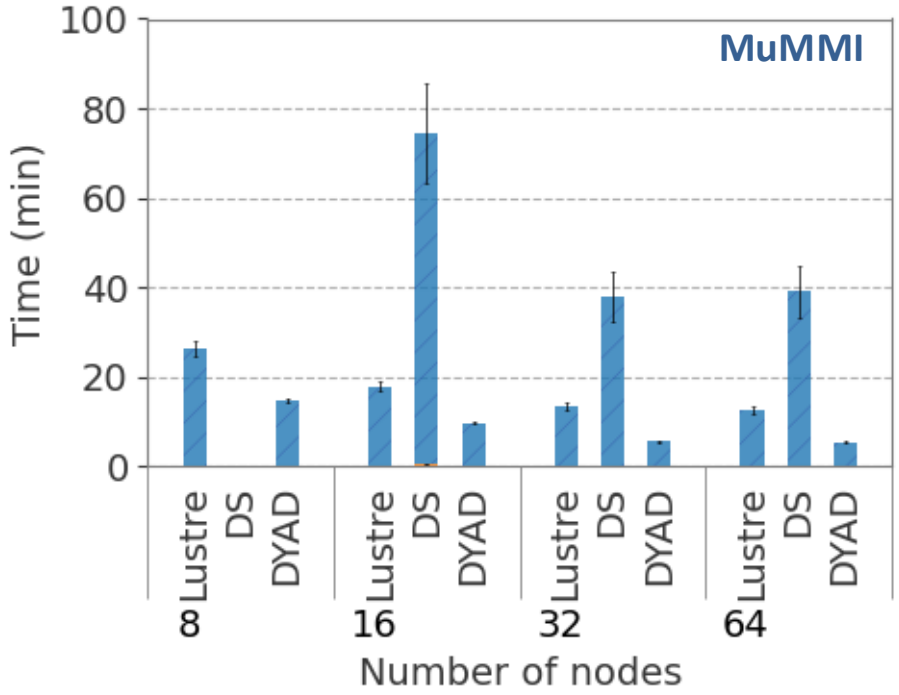
Hierarchical Sample Locator



Performance with distributed DL training

PyTorch data parallel training with 8 GPUs/node

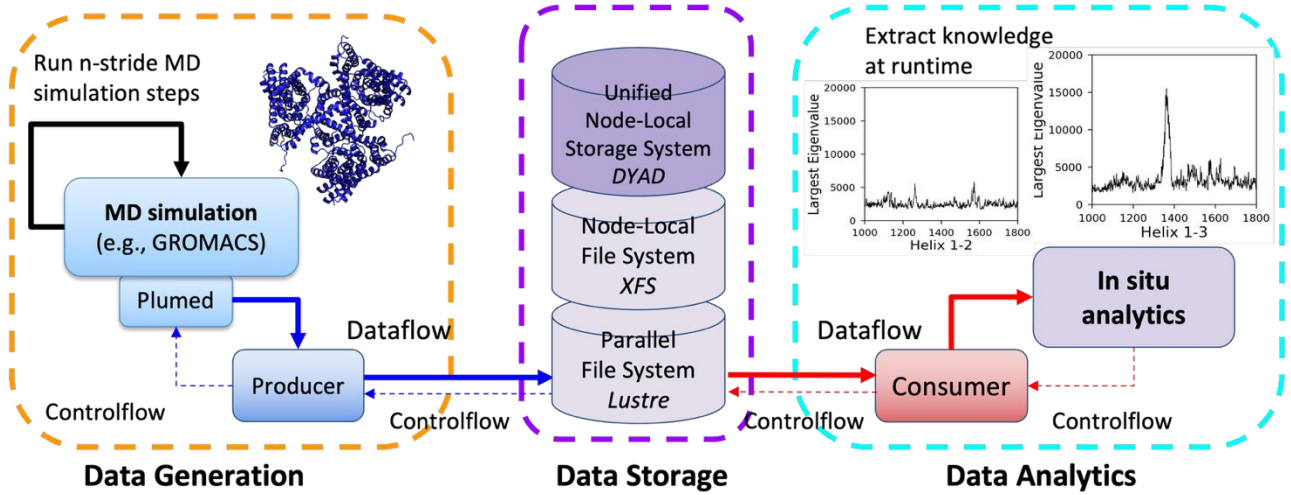




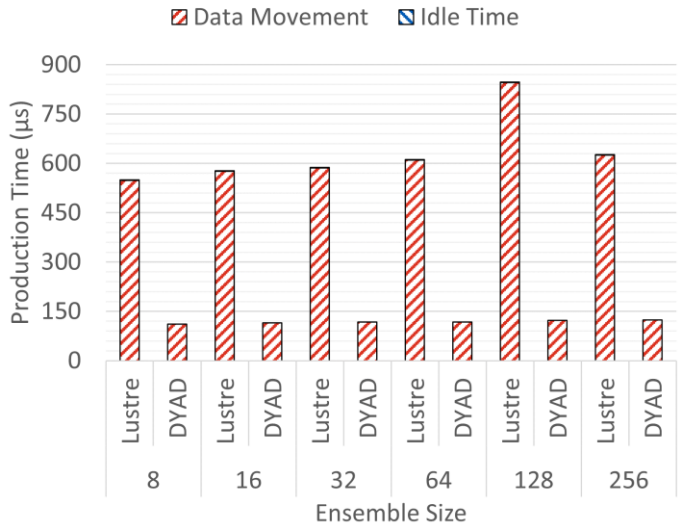
H. Devarajan, I. Lumsden, C. Wang, K. Georgouli, T. Scogland, J. Yeom, and M. Taufer, "DYAD: Locality-aware Data Management for Accelerating Deep Learning Training," IEEE Int'l Symp. on Computer Architecture and High Performance Computing (SBAC-PAD), Nov 2024.

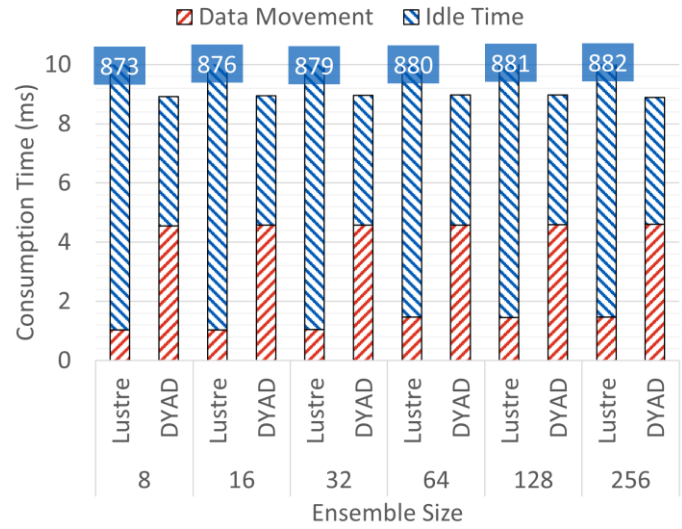
DYAD significantly improves (8.2x) the performance of I/O-bound distributed DL-training

Performance with *in situ* analysis workflow for MD simulation



8 processes/node





I. Lumsden, H. Devarajan, J. Marquez, S. Brink, D. Boehme, O. Pearce, J. Yeom, and M. Taufer, "Empirical Study of Molecular Dynamics Workflow Data Movement: DYAD vs. Traditional I/O Systems," IEEE Int'l Workshop on High Performance Computational Biology (HiCOMB), May 2024.

Conclusions

1. DYAD improves sample sharing for DL workloads by making use of their unique behavior and storage requirements.
2. DYAD's novel streaming RPC over RDMA protocol boosts inter-node access by 1.25-8.2x gain over state-of-the-art solutions.
3. DYAD data movement coordination strategy reduces network contention and thus improving data movement by 8x.
4. DYAD employs a hierarchical sample discovery technique to isolate metadata accesses, and that improves lookup throughput by 1000x at average.
5. DYAD optimizes large-scale DL workloads by 8.2x as compared to state-of-the-art solutions.

Publications

- H. Devarajan, I. Lumsden, C. Wang, K. Georgouli, T. Scogland, J. Yeom, and M. Taufer, "DYAD: Locality-aware Data Management for Accelerating Deep Learning Training," IEEE Int'l Symp. on Computer Architecture and High Performance Computing (SBAC-PAD), Nov 2024.
- I. Lumsden, H. Devarajan, J. Marquez, S. Brink, D. Boehme, O. Pearce, J. Yeom, and M. Taufer, "Empirical Study of Molecular Dynamics Workflow Data Movement: DYAD vs. Traditional I/O Systems," IEEE Int'l Workshop on High Performance Computational Biology (HiCOMB), May 2024.
- J. Yeom, D. H. Ahn, I. Lumsden, J. Luettgau, S. Caino-Lores, and M. Taufer, "Ubique: A New Model for Untangling Inter-task Data Dependence in Complex HPC Workflows," IEEE 18th Int'l Conf. on e-Science, Oct. 2022.
- https://github.com/flux-framework/dyad

Contributors

- Jae-Seung Yeom
- Hariharan Devarajan
- Chen Wang
- Dong Ahn
- Ian Lumsden
- Michela Taufer
- Jakob Luettgau
- Silvina Caino-Lores


Lawrence Livermore
National Laboratory