Lawrence Livermore National Laboratory

Enabling Transparent, High-Throughput Data Movement for Scientific Workflows on HPC Systems

THE UNIVERSITY OF TENNESSEE KNOXVILLE

CASC Center for Applied Scientific Computing

Ian Lumsden‡ (Student)

Jae-Seung Yeom*, Hariharan Devarajan*, Kathryn Mohror*, Michela Taufer‡ (Advisors)

* Lawrence Livermore National Laboratory, ‡Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville

GLOBAL COMPUTING LAB

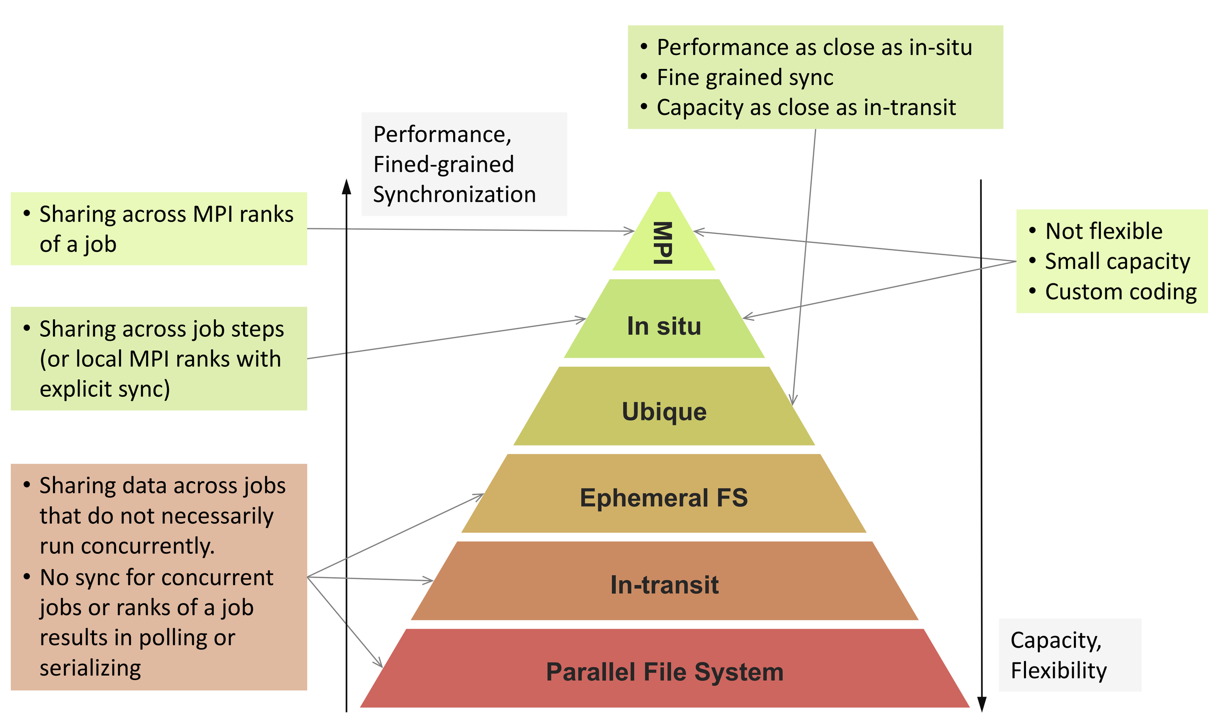
ABSTRACT

This poster presents the DYnamic and Asynchronous Data Streamliner (DYAD) middleware that provides an efficient and transparent method for data movement in scientific workflows based on the producer-consumer paradigm. We develop DYAD on top of Flux, a fully hierarchical HPC workload manager, and Unified Communication X (UCX), a unified framework for networking on HPC systems. We measure DYAD's performance with a suite of mini-apps and show how it outperforms traditional methods for data transfer while providing a high level of transparency.

CHALLENGES

There exists various approaches to resolve the **inter-task data dependence**, namely the sequential approach, which is based on a shared parallel file system (PFS), and the ***in situ*** approach [1]. However, both approaches retain one or more of the following major drawbacks:

- **Lack of synchronization support at the file/data object level**: requires workflow themselves to synchronize consumer and producer tasks to handle cases like a consumer task attempting to read a file before the producer completes its writing.
- **Poor temporal/spatial locality**: Workflows use coarse grained synchronization thereby a consumer task does not start its program execution before its dependent producer finishes its entire program execution, incurring distant temporal distance to resolve a data dependency, and each file travels a long spatial distance, missing bypass opportunities.
- **Low file metadata-operation performance**: Massive numbers of small files are often employed for emerging ML-based workflows, and hence the performance of file transfers is ultimately limited by the metadata performance of the PFS.
- **Conflict with code change requirements**: Many emerging workflows compose reusable components with minimum or no code change requirement on the pre-existing programs, and hence extensive changes needed to implement the aforementioned synchronization mechanism are often a nonstarter.



APPROACH

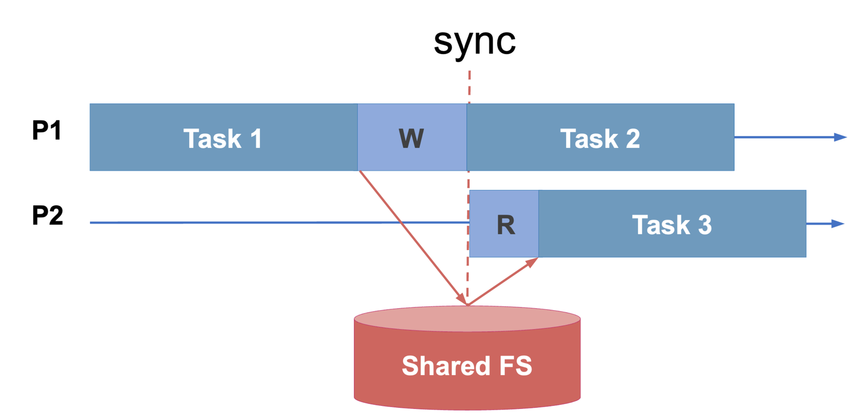
Problem: Couple an application that produces data with another that consumes the data with minimum or no code change.

```
void producer() {
   for (i=1; I <= N; i++) {
      produce(data[i])
      write(data[i]);
   }
}
void consumer() {
   for (i=1; I <= N; i++) {
      read(data[i]);
      consume(data[i])
   }
}
```

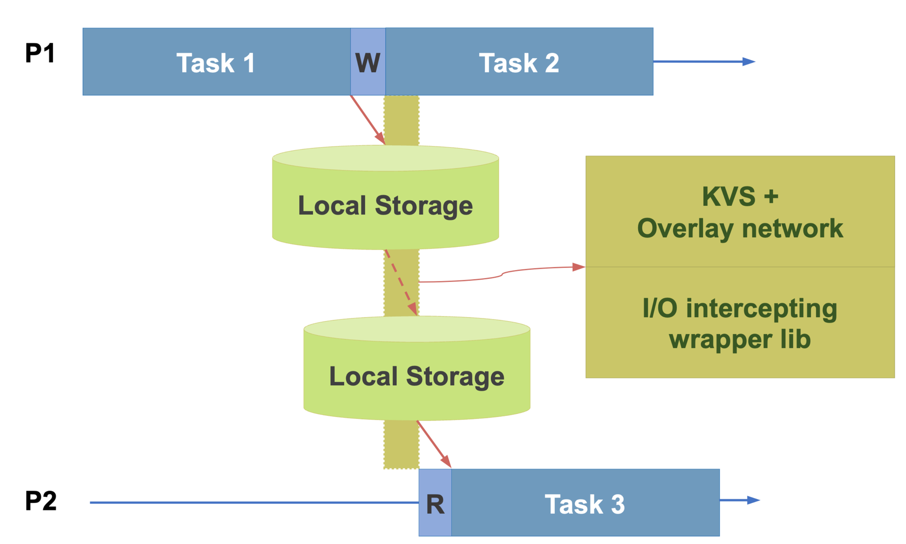
```
- description: producer task
  name: run-producer
  run:
    cmd: producer

- description: consumer task
  name: run-consumer
  run:
    cmd: consumer
    depends: [run-producer]
```

(a) Producer-consumer example (b) Maestro [2] specification in YAML

Solution 1: Sequential approach, as exemplified with Maestro, relies on a shared file system and an explicit synchronization between the end of the producer application and the start of the consumer.



Solution 2: Ubique approach [6] relies on local storages as well as transparent data transfer between storages and synchronization per shared file.



USER INTERFACE

- Running the dyad service with Flux [3]
  - `flux exec r all flux module load dyad.so /ssd/managed_dir`
- Running a user application with one of DYAD's user interfaces
  - Wrapper Library: replaces C file I/O calls with DYAD code using `LD_PRELOAD`
  - C++ Library: wraps C++ filestreams
  - Python Library: wraps Python's `open` function

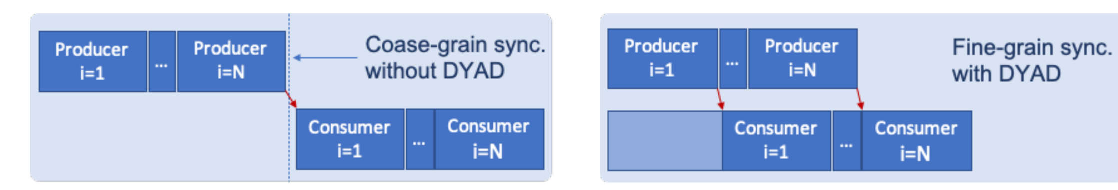
BENEFITS

No code change allows the benefits of

- Productivity (Fast construction of workflows with less effort)
- Easy debugging
- Portability (independent of any specific API other than widely used POSIX IO)

Performance benefits:

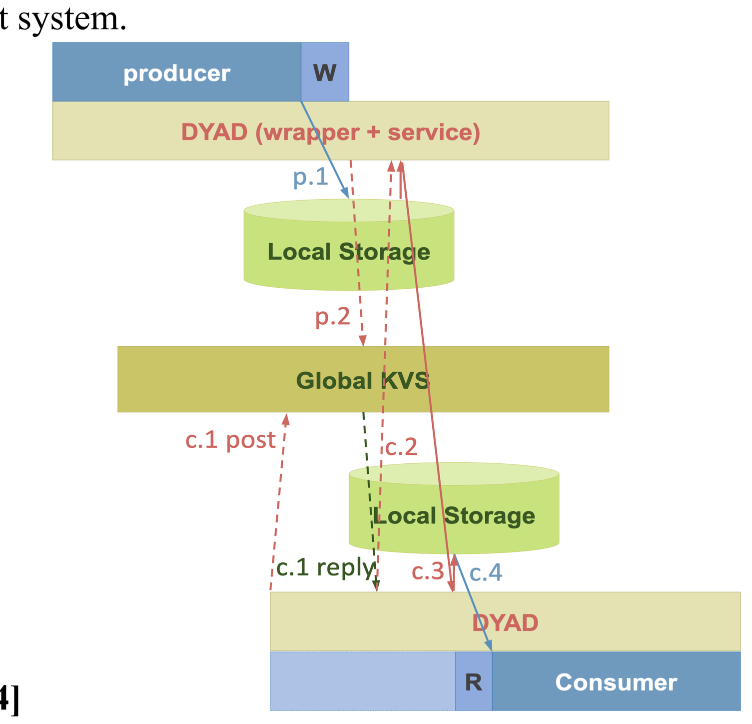
- Use of local storage enables faster accesses to storage and allows avoiding metadata operation bottleneck of PFS.
- Fine-grain file level synchronization with Ubique exposes further parallelism.



(a) Solution 1 **(b) Solution 2**

IMPLEMENTATION

DYAD: An embodiment of the Ubique model under Flux [3] resource and job management system.



DYAD [4]

- DYAD server runs on each node.
- DYAD client only intercepts I/O on files under the directory it manages.
  - If a file is on a local storage (LS), synchronize accesses and transfer it.
  - If it is on a shared storage, only synchronize accesses.

Producer

- p.1 *write(manged_dir/filepath)*
- p.2 *publish(<filepath, prod_rank>)*
  - If a file is written into ***managed_dir*** or its subdirectory, DYAD registers the filepath into the global key-value-store (KVS) of Flux.
  - KVS entry is a pair of ***filepath*** and ***prod_rank***, where prod_rank is the Flux rank of the service on the node where the producer is running on.

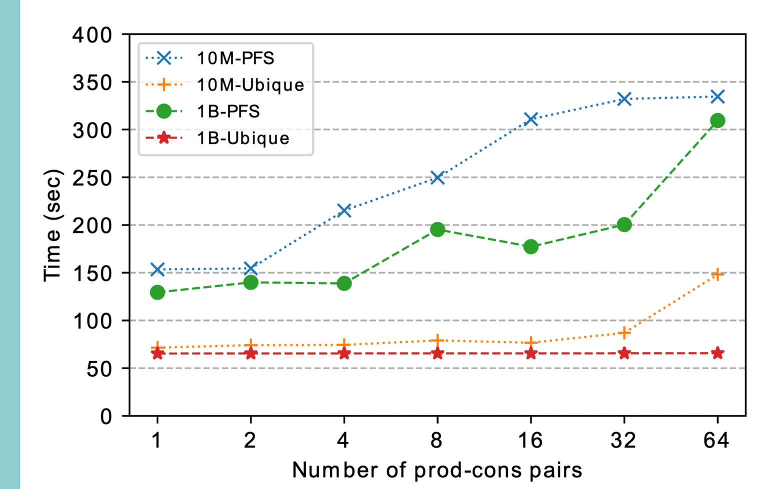
Consumer

- c.1 *query(filename)→ prod_rank*
  - Consumer queries KVS to obtain the rank of the file owner (producer). Then, blocking wait.
- c.2 *rpc_get(prod_rank, filename)*
  - Consumer asks the owner rank to transfer the file. DYAD module transfers the file to consumer using UCX. Once received, consumer stores it on LS
- c.3 make a copy of the data file on consumers LS
- c.4 *read(managed_dir/filename)*

GitHub Repo:

RESULTS

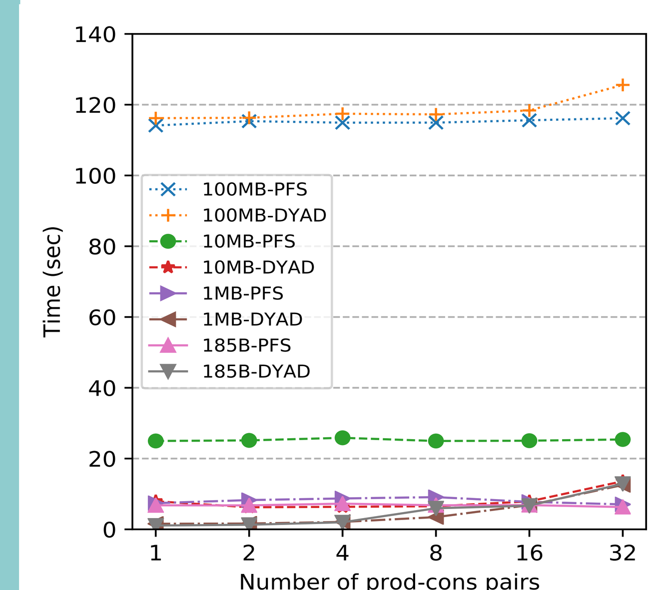
Synthetic Producer-Consumer Benchmark



Experimental setup:

- 1 second-long computation between file I/Os.
- 1 process (either producer or consumer) per node.
- Each producer-consumer pair exchanges 64 files.
- Measured on Quartz@LLNL: Intel Xeon E5-2695 v4 w/ 128 GB
- Node local storage of DYAD: tmpfs (memory)
- Parallel File System: Lustre

Molecular Dynamics-Inspired Producer-Consumer Benchmark



Experimental setup:

- Built on the Analytics4MD [5] framework
- Synthetic molecular dynamics data generation and analysis
- 1 process (either producer or consumer) per node.
- Each producer-consumer pair exchanges 64 files.
- Measured on Corona@LLNL: AMD Rome w/ 256 GB
- Node local storage of DYAD: on-node NVRAM
- Parallel File System: Lustre

CONCLUSIONS

- We present DYAD, a middleware for efficient and easy-to-use data file-sharing for scientific workflows based on the producer-consumer paradigm on HPC systems.
- We show that DYAD can speed up small- and mid-scale producer-consumer workflows up to six times compared to a sequential approach for moving data.
- In future work, we will examine DYAD's performance at larger scales and for a broader set of applications.

REFERENCES

[1] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: An interaction and coordination framework for coupled simulation workflows," in ACM Intl. Symp. on High Performance Distributed Computing, 2010.

[2] Maestro Workflow Conductor, https://maestrowf.readthedocs.io/en/latest/

[3] D. H. Ahn, N. Bass et al., "Flux: Overcoming scheduling challenges for exascale workflows," Future Generation Computer Systems, vol. 110, pp. 202–213, 2020.

[4] DYAD repository: https://github.com/flux-framework/dyad

[5] M. Taufer, S. Thomas et al., "Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-Generation Supercomputers," in IEEE eScience, 2019.

[6] J. S. Yeom, D. H. Ahn et al., "Ubique: A New Model for Untangling Inter-task Data Dependence in Complex HPC Workflows," in IEEE eScience, 2023.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344. LLNL-POST-852752