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

Ian Lumsden[1] (Student)
Jae-Seung Yeom[2], Hariharan Devarajan[2], Kathryn Mohror[2], Michela Taufer[1] (Advisors)
[1]University of Tennessee, Knoxville, TN, USA
[2]Lawrence Livermore National Laboratory, Livermore, CA, USA

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.

1 INTRODUCTION

Scientific workflows move data, often in the form of files, between tasks using the producer-consumer paradigm. This paradigm is challenging to perform optimally due to the complexity of spatial and temporal locality. For example, producer and consumer tasks can be either collocated on the same nodes or placed on entirely separate nodes depending on their resource requirements. Additionally, consumer tasks must be synchronized per-file or per-task due to the data dependence between producer and consumer. As a result, tasks can run concurrently or sequentially depending on the desired synchronization frequency. Another challenge is that existing data-sharing solutions may not be sufficient to support the optimal synchronization frequency and may depend on persistent data availability.

Two common data-sharing approaches in workflows aim to overcome these challenges: the sequential and in situ approaches. In the sequential approach, the consumer task runs after the producer while using file storage (e.g., Lustre) to share data between tasks. This approach is the most common due to its ease of use, but its sequential execution and slow shared file storage cause poor performance. In the in situ approach, the producer and consumer tasks run concurrently, sharing data using a dedicated middleware (e.g., DataSpaces [2]). This approach performs best due to its parallel execution and fast, resource-local storage solutions (e.g., in memory data stores). Nevertheless, implementations of the in situ approach usually require extensive modifications to workflow code.

To address these challenges, we propose DYAD, a middleware that combines the benefits of both approaches while providing a generality not found elsewhere (e.g., LowFive [3]). Our middleware efficiently handles producer-consumer data file-sharing in scientific workflows, providing performance and transparency. This poster has two contributions. First, we present DYAD's design, built on top of Flux [1], a fully hierarchical HPC workload manager, and Unified Communication X (UCX) [4], a unified framework for networking on HPC systems. Second, we design a mini-app suite, including a molecular dynamics (MD) workflow. We evaluate DYAD with our mini-app and compare its performance sequentially. This work shows that DYAD enables efficient data movement on HPC systems.

SC '23, November 12–17, 2023, Denver, CO
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 DESIGNING THE DYAD MIDDLEWARE

Designing DYAD's software components. DYAD uses a disaggregated server-client design [6]. The DYAD server is designed as a Flux broker module (i.e., callbacks invoked on the Flux broker) [1]. This allows the server to be distributed across the same compute nodes as the producer and consumer tasks without using large amounts of resources. The DYAD clients support workflow tasks through three interfaces: (1) a C interceptor library, (2) a C++ filestream wrapper, and (3) a Python module. DYAD's interfaces can be integrated into user code with minimal changes. All interfaces leverage a core library that abstracts the networking for control and data movement to and from the DYAD server. Figure 1 shows the DYAD server and client design.

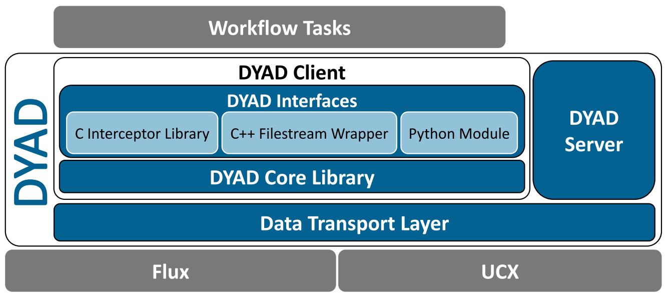

Figure 1: DYAD server and client design.

Augmenting the DYAD server and core library with UCX. To support different HPC systems, we design a communication abstraction on top of the networking tasks DYAD performs to move data. This abstraction is implemented as DYAD's data transport layer. The DYAD server and core library use the data transport layer to move data between producer and consumer. We implement this layer using UCX [4] to enhance DYAD's performance. Specifically, we use UCX's tag-matching two-sided communication. We choose this type of communication because it implements an MPI-style rendezvous protocol that automatically and transparently switches from two-sided communication to one-sided remote direct memory access communication based on data size.

Moving data with DYAD. Users start the DYAD server using Flux commands [1]. When starting their workflow tasks, users define

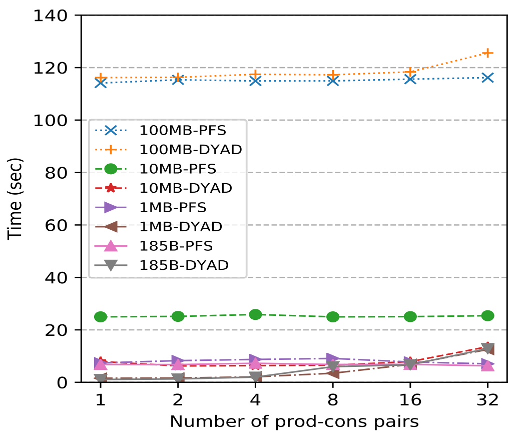

Figure 2: Times for the A4MD mini-app for different numbers of producer-consumer pairs and different file sizes.

either a producer-managed or consumer-managed directory. DYAD tracks these directories to trigger data production and consumption events. When a file in the producer-managed directory is closed, DYAD produces it by registering it in Flux's key-value store service. When a file in the consumer-managed directory is opened, DYAD consumes it by fetching it from the node on which it is stored using Flux and the DYAD server.

3 EVALUATING DYAD'S PERFORMANCE

Designing a testing performance suite. To evaluate the performance of DYAD, we design a suite of mini-apps emulating real scientific workflows. Our current suite includes a mini-app mimicking data movement in molecular dynamics (MD) simulations. Specifically, we design a plugin system that extends the Analyitcs4MD (A4MD) workflow [5] to run as a mini-app on top of DYAD. By combining these mini-apps with performance measurement tools (e.g., Caliper and HPCToolkit), we enable the in-depth study of DYAD's performance.

Analysis of mini-app performance. To evaluate the efficiency of DYAD, we test it with the mini-apps from our performance suite. We compare DYAD with a sequential approach, which is commonly used in scientific workflows. For this comparison, the success metric is the time between the synchronized start and end of all tasks in a mini-app. Figure 2 shows times from our A4MD mini-app for different numbers of producer-consumer pairs and different file sizes. The tests are executed on the Corona system at Lawrence Livermore National Laboratory, which has 48 AMD Rome cores, 256 GB of memory, and 1.5 TB of NVRAM storage per node. For each test, the producer-consumer pairs are distributed such that each task (i.e., producer or consumer) has exclusive compute node control. In each pair, the producer sends 64 fixed-size files to the consumer. Before each send, the producer generates the file on-the-fly using NumPy. As a result, the amount of work for a single pair is fixed. Since we scale the number of pairs, this test evaluates the performance of DYAD and the sequential approach under weak scaling. When moving data with DYAD, data is temporarily stored in the on-node NVRAM storage. When moving data with the sequential approach, data is stored in Lustre.

In the figure, we observe that DYAD outperforms the sequential approach for 185 byte, 1 MB, and 10 MB file sizes and small numbers of producer-consumer pairs. Specifically, we observe that DYAD runs up to 6 times faster than the sequential approach at these scales. As we increase the number of pairs, we also observe that DYAD's performance begins to scale linearly, while the performance of the sequential approach remains constant. Overall, DYAD outperforms the sequential approach for small and medium file sizes. However, it performs similarly to the sequential approach for larger files, suggesting that further optimizations are needed to ensure efficient scaling for large files. .

4 CONCLUSIONS

This poster presents 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 how our middleware 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.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-ABS-852835).

REFERENCES

[1] D.H. Ahn, J. Garlick, et al. 2014. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. In *Proceedings of the 43rd International Conference on Parallel Processing Workshops.*
[2] C. Docan, M. Parashar, and S. Klasky. 2010. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing.*
[3] T. Peterka, D. Morozov, et al. 2023. LowFive: In Situ Data Transport for High-Performance Workflows. In *Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS).*
[4] P. Shamis, M.G. Venkata, et al. 2015. UCX: An Open Source Framework for HPC Network APIs and Beyond. In *Proceedings of the 23rd IEEE Annual Symposium on High-Performance Interconnects.*
[5] M. Taufer, S. Thomas, et al. 2019. Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-Generation Supercomputers. In *Proceedings of the 15th International Conference on eScience (eScience).*
[6] J.S. Yeom, D.H. Ahn, et al. 2022. Ubique: A New Model for Untangling Inter-task Data Dependence in Complex HPC Workflows. In *Proceedings of the 18th International Conference on e-Science (eScience).*