Empirical Study of Molecular Dynamics Workflow Data Movement: DYAD vs. Traditional I/O Systems

Ian Lumsden*, Hariharan Devarajan†, Jack Marquez*, Stephanie Brink†,
David Boehme†, Olga Pearce†, Jae-Seung Yeom†, Michela Taufer*
*Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA
†Lawrence Livermore National Laboratory, Livermore, CA, USA

***Abstract*—This experimental work examines data movement in molecular dynamics (MD) workflows, comparing the Dynamic and Asynchronous Data Streamliner (DYAD) middleware with traditional, industry-standard I/O systems such as XFS and Lustre. DYAD moves MD simulation frames to analytics processes, providing enhanced flexibility and efficiency for dynamic data transfers and in situ analytics. At the same time, traditional I/O storage systems provide durability and scalability for high-performance computing (HPC) systems. The study integrates MD workflows with common simulation codes, facilitating immediate capture and transfer of MD frames to a staging area. It explores various molecular models, from simple to complex, assessing data management performance and scalability. Different producer-consumer pairs, molecular models, and data transaction frequency enable testing across small to large-scale HPC scenarios, from single-node configurations to large, distributed environments. The findings reveal that adaptive mechanisms for minimizing synchronization, direct network communication between producer and consumer processes, and optimizations of both data movement and synchronization are crucial for performance and scalability in MD workflows.**

***Index Terms*—Producer-Consumer Paradigm, Molecular Dynamics Workflows, Dynamic Data Management, High-Performance Computing, In Situ Analytics**

I. Introduction

This experimental work characterizes the data movement in molecular dynamics (MD) workflows, particularly focusing on the movement of MD-generated data during runtime. The study evaluates the use of an advanced software middleware called the Dynamic and Asynchronous Data Streamliner (DYAD) [1] versus traditional, industry standard I/O systems: node-local file systems like XFS [2] and parallel file systems like Lustre [3]. All three data management solutions deal with the data movement needs within MD workflows from distinct angles. With its advanced abstractions, DYAD enhances flexibility and efficiency in handling dynamic data movement and automated workflow synchronizations. XFS provides the ability for efficient in situ analytics before data is moved off node. Lustre's durability and scalability make it suitable for high-performance computing (HPC) applications that require highly parallel data access over a network of nodes. We focus on the movement of MD-generated data (i.e., moving frames from the simulations to the analytics) rather than on the MD process itself (i.e., efficient computation of molecular interactions, parallelization, or GPU acceleration).

We study MD workflows essential for capturing the behavior of molecular models, coupling data movement of MD simulations with in situ analytics. The workflows seamlessly integrate with prevalent MD simulation codes [4]–[6], thereby facilitating the immediate capture of MD frames in memory during runtime. These frames are then efficiently transferred to a staging area, leveraging the advantages of both advanced software middlewares and traditional I/O systems. We examine a spectrum of molecular models with varying size and complexity, ranging from JAC [7] (a simple 23,558 atom, 644 KiB molecular model) to STMV [8] (a complex 1,066,628 atom, 28.5 MiB model). By scrutinizing the data handling of various models, we thoroughly evaluate the performance and scalability of data management solutions in MD workflows.

We adjust multiple parameters within the MD workflow to test the data management systems' capabilities. We vary the number of producer-consumer pairs, reflecting different parallelism scales across our experiments. We test four different molecular models with an increasing number of atoms, thus generating different frame sizes. We manipulate the stride of data movement, which influences the frequency of data transactions. Our tests enable us to mimic a range of realistic scenarios that could be encountered in HPC environments, from small, single-node operations to large, distributed computations across multiple nodes. Ultimately, our comprehensive approach allows us to identify the strengths and limitations of current technologies and offers a guide for future development in MD simulation data management.

We identify five main findings from our empirical study. First, prioritizing a data management system with adaptive synchronization mechanisms can lead to substantial overall scalability for MD workflows on a single-node configuration, even if there is also a small increase in production time. Second, leveraging network communication for data movement in small-scale distributed MD workflows has little effect on performance when communication occurs directly between two nodes. Third, selecting a data management system that optimizes both data movement and synchronization ensures overall performance in a large-scale, distributed MD workflow. Fourth, leveraging local resources and efficient communication protocols enables better scalability as data sizes increase. Finally, minimizing synchronization is critical when the frequency of data transfer between producers and consumers decreases.

II. MD Workflows: Simulation and Analytics

Modern MD workflows are increasingly sophisticated and involve a combination of simulation and real-time, in situ analytics. These workflows have producer-consumer patterns. The MD simulation produces frames (the atom list and their 3-D locations) at a regular number of steps (i.e., strides), and the in situ analytics consume those frames.

A. MD Simulations

MD simulations have been a cornerstone in computational science. They are among the most frequent workloads on exascale machines [9]. These simulations allow scientists to observe atomistic details of biological processes that are often elusive to experimental techniques. The complex interaction of atoms and molecules is at the heart of MD simulations. Each simulation replicates the behavior of a molecular model by using a two-step algorithm. The process begins with calculating interatomic forces using force fields (a mathematical representation of the energy landscape dictating atomic interactions). After force calculation, the simulation updates the positions of atoms by Newton's equations, marching forward in time through small increments. Models comprising hundreds of thousands of atoms require large computing power executed on GPUs.

A large-scale MD simulation is an ensemble of MD jobs (as many as hundreds of thousands) that run on different compute nodes and produce independent trajectories [10]; each job simulates the same molecular model starting from different initial conditions (e.g., positions, velocities) or similar models under different conditions (e.g., temperature, protein mutants, drug variants). Each MD job reproduces the evolution of the relevant molecular model by computing and writing to storage the model's atomic coordinates (frame) and other relevant properties at regular intervals as the job evolves in time. The sequence of molecular conformations (the trajectory) is written to disk.

B. In Situ Analytics

Exascale systems increase the capabilities of MD simulations. The data generated from these simulations are large and provide an even finer granularity of molecular detail. However, data management and real-time analytics are challenging for researchers who study the data as it is generated to steer the simulation (e.g., terminate or fork a trajectory) and annotate the events for retrieval and visualization [10].

Run-time, in situ analytics in MD workflows refers to data analysis as the data is generated during simulations, without requiring data to be written to disk and analyzed post-process. This approach is particularly beneficial for large-scale simulations where the volume of generated data is immense, making traditional post-processing not only cumbersome but also time-consuming and resource-intensive [11]. Adaptive simulations leverage in situ analytics to concentrate resources on significant phenomena for more effective exploration. Run-time, in situ analytics techniques include monitoring, data reduction, advanced visualization, and on-the-fly analysis [12].

Real-time monitoring enables researchers to observe simulation progress and identify phenomena as they occur, aiding in troubleshooting. Data reduction techniques within in situ analytics streamline data management by storing only crucial information to minimize storage needs. Advanced visualization offers immediate insights into simulations, allowing scientists to witness interactions dynamically [13]. On-the-fly analytics process simulation data in real-time for property and metric calculations, eliminating the need to store entire datasets [14].

III. Data Management Solutions

Integration of MD workflows and HPC is essential for managing computational demands and ensuring efficient data processing. We explore two distinct approaches for data transfer: an advanced middleware with abstractions like the Dynamic and Asynchronous Data Streamliner (DYAD) and two traditional I/O solutions: node-local file system with XFS and parallel file system with Lustre. Figure 1 shows the MD workflow with the three different data management solutions. The figure shows the MD workflow from simulation to in situ

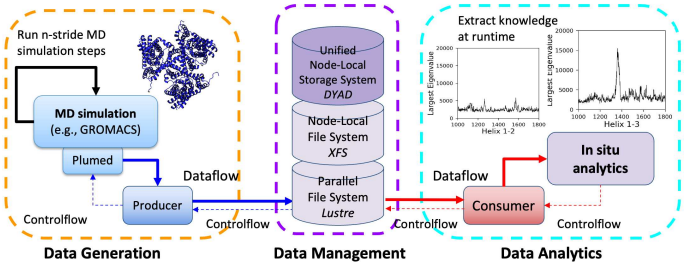


Fig. 1: MD Workflow from simulation to in situ analytics using three different data management solutions (i.e., DYAD, XFS, and Lustre).

analytics. On the left side of the figure, an MD simulation (e.g., using GROMACS [4]) is enhanced by Plumed [15] to capture the MD frames without disrupting the execution. MD frames are fed into a producer. The producer then transfers data into one of three possible data management solutions (i.e., a node-local storage-based middleware with DYAD, a node-local file system with XFS, or a parallel file system with Lustre). On the right side of the figure, a consumer extracts data from the management systems for in situ analytics at runtime. The figure presents an example of in situ analytics of three interacting secondary structures (i.e., Helix 1-2 and Helix 1-3) in [12]) with two graphs depicting the largest eigenvalues of two of the three helices within an MD simulation and respecting sudden changes in the molecular model.

An important aspect of the data movement illustrated in Figure 1 is synchronization. Since producers and consumers typically require significant resources, they are often not located on the same node, necessitating a deliberate synchronization barrier to manage the transition from production to consumption. The time associated with this synchronization consists of two parts. The first part of synchronization time consists of a delay for the producer to generate a piece of data. The

remaining synchronization time consists of the time needed for the consumer to discover that the data has been produced.

A. Advanced Middleware: DYAD

DYAD [1] provides a sophisticated approach to data movement and management. It is designed to optimize data transfer efficiency and enables high-performance data exchange between producer-consumer tasks within a workflow without the need for manual synchronization between these tasks. DYAD enables faster producer-consumer data exchange

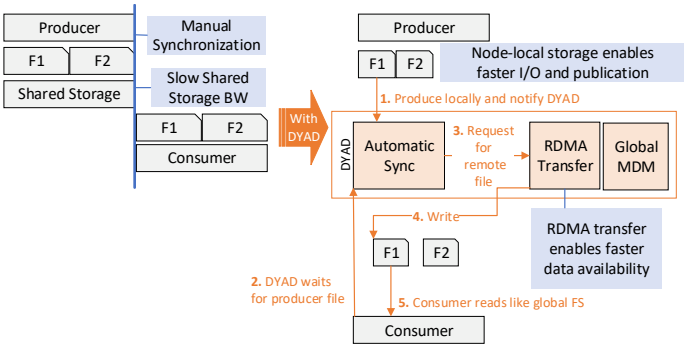


Fig. 2: DYAD enables faster producer-consumer data exchange by using node-local storage accelerators, multi-protocol automatic synchronization primitives, global metadata management, and RDMA-enabled data transfer.

by using node-local storage accelerators, multi-protocol automatic synchronization primitives, global metadata management (MDM), and RDMA-enabled data transfer.

Figure 2 illustrates a producer-consumer workflow utilizing shared storage, comparing scenarios without and with DYAD integration. On the left, the diagram shows the traditional workflow without DYAD, relying on manual synchronization for data exchange between producers and consumers. This synchronization usually involves MPI primitives, file system polling in workflow managers like Pegasus [16], or file system locks, with the method chosen based on the system's architecture, the storage type, and the available data exchange protocols [1]. The complexity of this synchronization leads many workflows to adopt a coarse-grained approach, which, while simplifying the design, extends the overall duration of the workflow by not overlapping producer and consumer tasks. On the right, the figure shows a workflow incorporating DYAD for data exchange, highlighting three key features. First, DYAD eliminates the requirement for manual synchronization between tasks. Through DYAD's automatic synchronization, producers and consumers can run in a concurrent and pipelined manner, and the consumer will automatically wait for data to be made available. This pipelinning reduces the cost of delay in synchronization. Second, DYAD leverages node-local accelerators, such as burst buffers, instead of relying solely on shared storage, significantly enhancing I/O bandwidth and scalability. Last, the transfer of data between producers and consumers is conducted through the efficient RDMA protocol, optimizing both the bandwidth available for consumption and the scalability of the workflow. DYAD's features collectively reduce the total duration of workflow tasks and improve the efficiency of modern HPC storage architectures without necessitating changes to the existing workflow structure.

The DYAD middleware supports dynamic data routing, asynchronous communication, and on-the-fly processing capabilities. Because of these features, the middleware is particularly beneficial in scenarios where the data generation rate varies significantly or when the data requires pre-processing or filtering before being stored or analyzed. Its dynamic nature enables an adaptive response to the computational load, network bandwidth, and storage capabilities, thereby minimizing bottlenecks and ensuring a smoother data flow.

B. Traditional I/O Solutions: XFS and Lustre

We examine two prevalent traditional I/O solutions used for data management in MD workflows: the node-local file system with XFS and Lustre. The node-local file system, implemented with XFS, offers a specialized setup where each node within a network possesses an exclusive file system. This arrangement enhances direct disk access for each node, markedly reducing access latency and interference while boosting I/O throughput, which is crucial for operations demanding quick data access. XFS enhances this framework by handling large files alongside journaling features that ensure data integrity, making it a scalable, high-performing, and dependable storage option. While it may lack the intricate abstractions of systems like DYAD, XFS still provides benefits in terms of performance and reliability. However, XFS is not capable of moving data between nodes. As a result, processes that move data through XFS must be collocated on the same node, which limits workflow scalability.

On the other hand, Lustre presents an easy-to-use solution for data transfer, renowned for its use in distributed parallel file systems that facilitate high-throughput access to extensive datasets across many nodes in HPC settings. Designed to support vast cluster computing endeavors, Lustre can manage data on the scale of petabytes and accommodate tens of thousands of client nodes. Though it does not feature the advanced abstractions of DYAD, Lustre provides remarkable scalability, performance, and reliability for large bulk-synchronous data movement. Its distributed architecture significantly improves the efficiency of data movement, allowing numerous clients to access and process data simultaneously, which makes Lustre an ideal choice for applications with large bulk-synchronous data movement that require significant bandwidth and massive data access.

Neither XFS nor Lustre provide any form of automated synchronization for producer-consumer data transfer. As a result, users of these file systems must implement synchronization manually. As previously mentioned, the complexity of this manual synchronization leads to the adoption of coarse-grained approaches. Although easy to implement, these approaches result in serialized execution of the producer and consumer, which increases the cost of delay in synchronization.

[1]The code is publicly available at https://github.com/flux-framework/dyad.

IV. Performance Study of Molecular Structures

We investigate MD simulations' performance and data management challenges across four distinct molecular structures, highlighting their data transfer performance and scalability. We collect key insights into the efficiency and scalability of different data management solutions (DYAD, XFS, and Lustre) through a detailed performance analysis in an MD-inspired workflow. The findings emphasize the importance of fine-grained synchronization and storage setup in optimizing MD simulations, offering guidance for selecting appropriate data management solutions based on molecular structure size and simulation complexity.

A. Molecular Models

Our tests use four distinct molecular models, each selected for its unique role in biological processes and suitability for data movement study. These models exemplify the diverse data volume generated in MD simulations (Figure 3). As the model size increases in the number of atoms, the MD simulation performance measured in nanoseconds per day decreases, requiring supercomputers to simulate larger models. We begin with JAC [7], a molecular structure comprising 23,558 atoms; JAC is frequently used as a standard model for probing protein folding mechanisms and dynamics within MD simulations. Next, we consider ApoA1 [17], consisting of 92,224 atoms, notable for its vital function in lipid transport and metabolism, particularly as a critical structural element of high-density lipoproteins (HDL) in circulatory systems. We then move to the F1 ATPase molecular structure [18], with 327,506 atoms, a key component in cellular respiration and ATP synthesis. Last, we use STMV [8], an extensive molecular structure with 1,066,628 atoms. STMV is recognized for its size and role as a satellite tobacco mosaic virus, providing a rich framework for studying the intricacies of virus assembly and protein-RNA interactions.

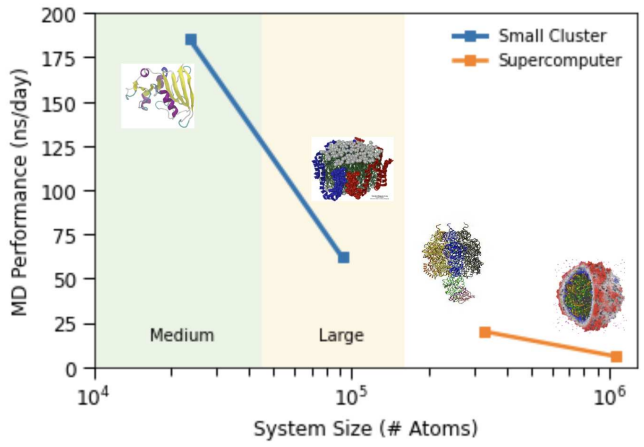


Fig. 3: Molecular structures with increasing size in atoms and associated MD frames generated for in situ analytics.

Tables I and II present the detailed setup of the four molecular structures used in our tests, each with various attributes relevant to MD workflows. These tables provide an overview of the molecular characteristics within their respective MD simulations. In Table I, the molecular structures are characterized by the number of atoms in each molecule, the estimated size of each frame in the MD simulation, and the computational performance measured in steps per second. We derive the steps per second values from ns/day and simulation configurations presented in [19]. For example, the JAC model has 23,558 atoms, and each frame is approximately 644.21 KiB in size, with the model achieving 1072.92 steps per second. The models vary in complexity. The STMV model is the largest; it contains over a million atoms and generates frames that are 28.48 MiB in size, but it runs at the slowest steps per second rate. Table II focuses on the stride of each molecular structure, a parameter that defines the sampling or output frequency within the MD simulation. Together with the steps per second, the table adds the time in milliseconds taken for each MD step. We derive strides for each model from [19] and [20] such that the frequency of data generation is equal for each molecular model.

TABLE I: Targeted molecular models [7], [8], [17], [18].

Name	Num Atoms	Frame size	Steps/second
JAC	23,558	644.21 KiB	1072.92
ApoA1	92,224	2.46 MiB	358.22
F1 ATPase	327,506	8.75 MiB	115.74
STMV	1,066,628	28.48 MiB	34.14

TABLE II: Stride for each molecular model [7], [8], [17], [18].

Name	Steps/second	ms/step	Stride	Frequency (s)
JAC	1072.92	0.93	880	0.82
ApoA1	358.22	2.79	294	0.82
F1 ATPase	115.74	8.64	92	0.82
STMV	34.14	29.29	28	0.82

B. Data Management Solution Configurations

Figure 4 presents three configurations of an MD workflow studied in this paper, each using a different data management solution: DYAD, XFS, or Lustre. Each configuration demonstrates a different approach to managing and transferring data within an MD workflow, from a single-node, tightly-coupled system to multi-node, distributed systems, highlighting the adaptability of MD workflows to various management solutions depending on computational and data requirements. In the first configuration, the workflow is contained within a single node that employs either the DYAD client or XFS. We consider DYAD and XFS as XFS is the fastest local storage solution we can deploy for such a configuration. The MD simulation, running on one core, passes frames directly to the in situ analytics on another core. This local exchange utilizes Solid-State Drive (SSD) storage, with control messages facilitating the coordination between the two processes. The second configuration spans two nodes but uses DYAD for data management. Here, Node 1 with the MD simulation and Node 2 with the in situ analytics each include a DYAD client, interfacing with a DYAD server that manages the data

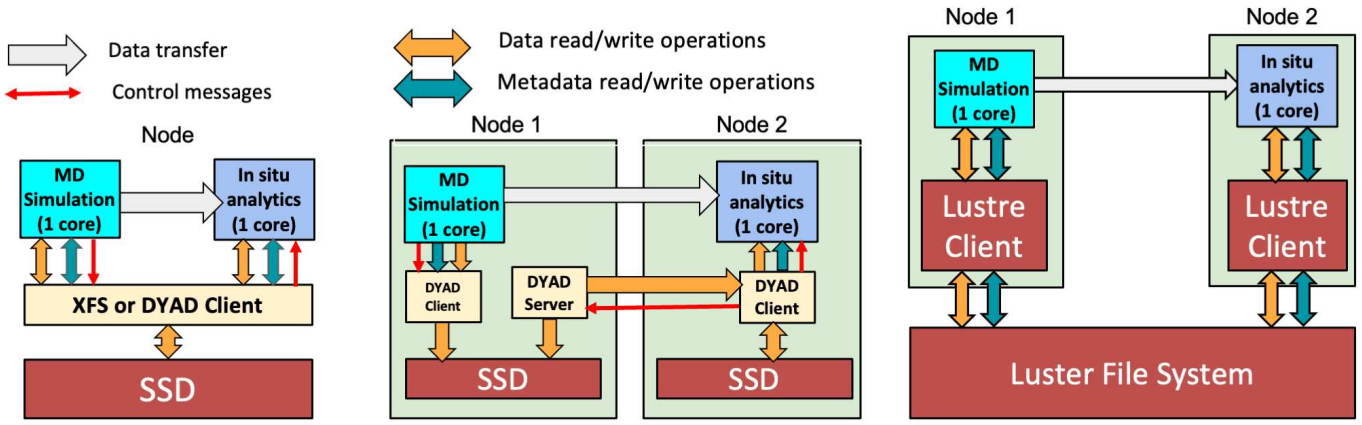


Fig. 4: MD workflow management solutions: single-node DYAD/XFS, multi-node DYAD, and multi-node Lustre configurations.

flow. This setup is designed to utilize DYAD's direct memory access capabilities, allowing for efficient data transfer and staging directly between nodes' local SSD storage, potentially bypassing some of the overhead associated with traditional file systems like Lustre. The third configuration spreads the workflow across two nodes via Lustre. Node 1 runs the MD simulation, and Node 2 runs the in situ analytics on their respective cores. Both simulation and analytics interact with Lustre clients running on Nodes 1 and 2, respectively. These clients communicate with a centralized Lustre file system that manages data storage and transfer. This setup allows for distributed computing where the MD simulation and analytics can happen on separate hardware, relying on Lustre's robust distributed file system to manage data coherence and availability.

C. Testing Platform and Setups

To produce a controlled environment for evaluating DYAD, XFS, and Lustre, we design and implement a point-to-point MD-inspired workflow similar to the one shown in Figures 1 and 4. Our workflow consists of an equal number of producer processes and consumer processes. We link individual producer and consumer processes through our data management solutions to create an ensemble of concurrently running pairs. Producer processes emulate MD simulation, while consumer processes emulate MD analytics. Each producer process runs for a fixed number of steps (i.e., stride) before producing a snapshot (i.e., frame). For each step, a producer emulates the computation done by an MD simulation using a fixed-duration MD sleep. A producer serializes the generated frame between each stride and writes it to DYAD, XFS, or Lustre using POSIX APIs. Each frame has a fixed number of atoms correlating to the molecule shown in Table I. To simplify the evaluation of these management solutions, we run every producer in our workflow for a fixed number of strides; this results in a fixed number of data transfers per producer-consumer pair. Each consumer runs for a fixed number of iterations that is equal to the total number of data transfers per producer-consumer pair. In each iteration, a consumer first reads a frame from DYAD, XFS, or Lustre using POSIX APIs. It then deserializes that frame before running a sleep to emulate the computation performed in data analytics. We set the duration of this sleep to match the frequency at which producer processes generate frames.

Our workflow can run either on a single node or across multiple nodes. All producer and consumer processes are collocated when running on a single node, using either the DYAD or XFS configuration shown in Figure 4. When running across nodes, we place only one process type per node (producers or consumers). As a result, our total number of nodes is evenly split between nodes running producer processes and nodes running consumer processes. This splitting causes our ensemble to use the multi-node DYAD or Lustre file system configurations shown in Figure 4. In both single-node and multi-node scenarios, we only place up to 8 processes per node because we only have 8 GPUs per node; in a real-world MD workflow, each simulation and analytics process would be tied to one GPU.

To evaluate the performance of DYAD, XFS, and Lustre, we consider three types of scaling in our workflow. First, we scale the size of the ensemble in terms of the number of producer-consumer pairs. Due to our 8-process per node limit, this scaling also requires us to scale the number of nodes. As a result, we perform three tests to examine ensemble size scaling: (1) a single-node test, (2) a two-node test, and (3) a multi-node test that scales from 2 to 64 nodes. Second, we scale the size of the molecular model (shown in Table I). When considering molecular size scaling, we also vary the stride so that our frequency of data generation is the same regardless of the molecular structure. The stride values used for each molecule are shown in Table II. Third, we scale the frequency of frame generation. We consider 1, 5, 10, and 50 strides to scale frequency for both the JAC and STMV molecular models, relying on strides deployed in [12]. For all our tests, we run each configuration 10 times.

For each run of each configuration, we collect performance data using Caliper [21]; we analyze that data using Thicket [22] and its query language [23]. Through these performance tools, we identify two components of data

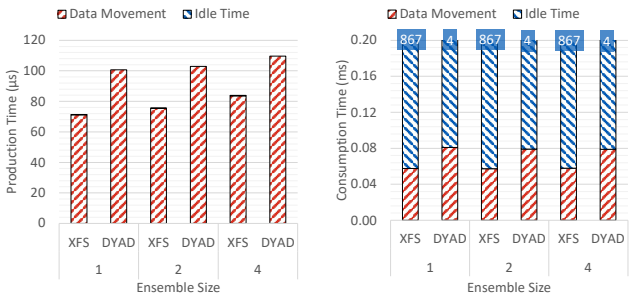
production and consumption time: data movement time and idle time. We define data movement time as the time to write data to or read data from our management solutions. For DYAD, we measure data movement time by recording the time spent in POSIX APIs and then subtracting away the idle time from lower levels of the software stack using Thicket. For XFS and Lustre, we measure data movement time by simply recording the time spent in POSIX APIs. We define idle time as time spent for synchronization. For DYAD, we measure idle time by recording the time spent in different components of DYAD and then isolating the components associated with synchronization using Thicket. For XFS and Lustre, we measure idle time by simply recording the time spent in a `MPI_Barrier` between each producer and consumer. We run all of our tests on Lawrence Livermore National Laboratory's Corona system [24]. Corona contains 121 compute nodes. Each node has 1 AMD EPYC 7401 CPUs and 8 AMD MI50 GPUs. Each node also has a 3.5 TB NVME SSD, which we use for XFS and DYAD in our testing. Nodes are connected by an InfiniBand QDR interconnect.

D. Ensemble Size Scaling

Through a series of three tests, we evaluate ensemble size scaling by comparing the performance of DYAD, XFS, and Lustre as we increase the number of producer-consumer pairs. We examine the effects of concurrent data movement on the performance and scalability of our data management solutions. Our comparison spans various configurations, including using a single node with local storage, a dual-node setup, and multiple-node pairs that progressively increase the number of producers and consumers. For each test, we move frames from MD simulations to in situ analytics, studying JAC with a fixed stride of 880 steps. Within each producer-consumer pair, we run the producer for 112,640 steps to produce 128 frames, and similarly, we run the consumer for 128 iterations to match the number of JAC frames created. This mimics MD simulations with a fixed stride of 880 steps, where the stride defines the frequency at which data is written and read.

In the first of the three ensemble size scaling tests, we measure the production and consumption times on a single node when using DYAD and XFS. By using a single node, DYAD does not use its network components. To produce a fair comparison, we choose to exclude Lustre from this test because POSIX-compliant parallel file systems, like Lustre, must move data across the network to remote storage in this type of workflow. We scale the size of the ensemble in terms of the number of producer-consumer pairs from 1 to 4. We only scale up to 4 pairs because there are only 8 GPUs per node on Corona, and each pair requires 2 GPUs: one for the producer and one for the consumer.

Figure 5(a) displays the data production time in microseconds for ensemble sizes of 1, 2, and 4 pairs. The figure depicts data movement time (red striped bars) and idle time (blue striped bars) for each ensemble size. We observe that adding more concurrent ensembles linearly increases the time for both DYAD and XFS. As the number of ensembles increases,



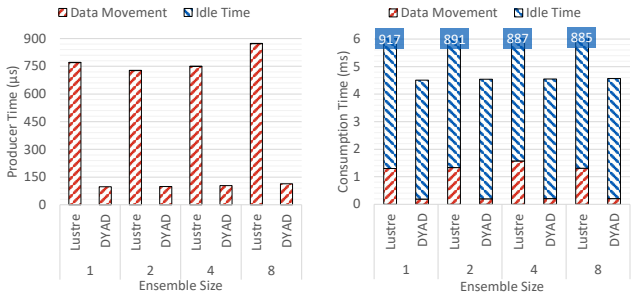
(a) Producer Performance (b) Consumer Performance

Fig. 5: Performance on a single node with JAC shows that (a) DYAD production time is 1.4 times slower than XFS due to global namespace management; (b) DYAD consumption time is 192.9 times faster than XFS due to lower idle time.

DYAD has 1.4 times more data movement cost than XFS due to DYAD's requirement to manage additional metadata, resulting in higher production time than XFS. There is no significant idle time in either solution.

Figure 5(b) illustrates the data consumption time in milliseconds under the same settings as Figure 5(a). The figure compares the time for data movement and idle time across the ensemble sizes of 1, 2, and 4 pairs for both DYAD and XFS. Similar to data production time, DYAD's data movement is 1.4 times slower than that of XFS. Unlike data production time, idle time affects data consumption time in both DYAD and XFS. In DYAD, the effect of idle time is limited; in XFS, the effect of idle time is significant. This difference in the effect of idle time is due to differences in synchronization of the producer and consumer. DYAD uses multi-protocol synchronization. For the first data consumption step, DYAD uses a loosely coupled synchronization based on a key-value store (KVS) because the data is not immediately available. This loosely coupled synchronization causes the consumer to wait for data availability, but it does not require the producer to wait on the consumer. As a result, the producer can continue with its next MD simulation step while the consumer reads the data. This overlap of the producer and consumer causes data to be available before every subsequent consumption step. When data is already available, DYAD automatically switches to a much less costly file lock-based synchronization. Due to this multi-protocol synchronization, DYAD will only incur the cost of its most expensive form of synchronization during the first consumption step, resulting in lower idle times than XFS. On the other hand, XFS uses a tightly coupled synchronization, where the producer waits for the consumer to begin reading data. This waiting causes the next consumption step to slow.

In summary, while data production with DYAD is 1.4 times slower than XFS due to the overhead of metadata management, DYAD significantly outperforms XFS in data consumption times owing to its lower synchronization costs; DYAD is 192.9 times faster than XFS in overall data consumption. Despite DYAD's marginally slower performance in data movement due



(a) Producer Performance (b) Consumer Performance

Fig. 6: The JAC molecular model on two nodes demonstrates a) DYAD is 7.5 times faster than Lustre for data production due to the use of node-local storage and b) DYAD is faster than Lustre by 6.9 times for consumer data movement, and, overall, DYAD is 197.4 times faster than Lustre.

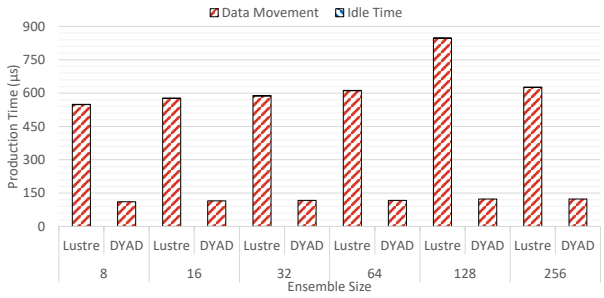
to additional metadata management, its efficiency in handling idle times, facilitated by finer-grained synchronization, considerably enhances the overall workflow efficiency.

> **Finding 1:** Prioritizing a data management solution with adaptive synchronization (despite a small increase in production time) can lead to substantial overall scalability for MD workflows on single-node configurations.

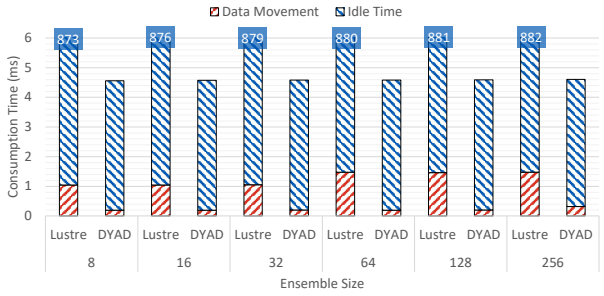
In the second ensemble size scaling test, we expand our measurement to a small-scale distributed environment with a two-node setup, allocating one node to producer processes and one node to consumer processes while utilizing DYAD and Lustre. We use Lustre for this test instead of XFS because XFS cannot run across multiple nodes. In Figure 6, we observe the times for data movement and idle times for writing JAC frames across ensemble sizes of 1, 2, 4, and 8 producer-consumer pairs, mimicking the time in the single-node configurations for DYAD. Introducing network communication for data movement with a small number of nodes does not negatively affect the scalability of DYAD. This suggests that the network overhead introduced by DYAD in a two-node distributed environment is manageable, and DYAD's performance remains consistent even when the data management tasks are spread across the two nodes. These results mirror observations in [12]. We also see that Lustre on two nodes is slower than XFS on one node (see test above) due to moving data to the off-node Lustre file system. DYAD's data movement in the producer is 7.5 times faster than that of Lustre because DYAD uses faster node-local storage. Similarly, in the consumer, we observe that DYAD's data movement is 6.9 times faster than Lustre.

> **Finding 2:** Leveraging network communication for data movement in small-scale distributed MD workflows has little effect on performance when the communication is direct between nodes.

In the final ensemble size scaling test, we further expand the scale of our distributed environment, and we measure the



(a) Producer Performance



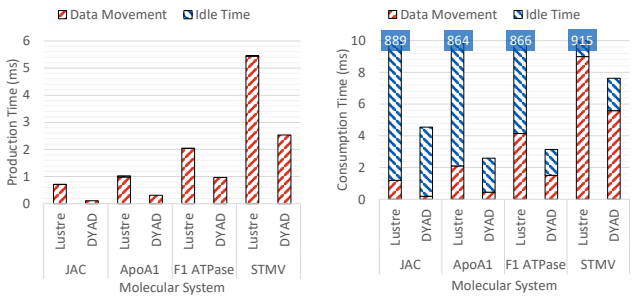
(b) Consumer Performance

Fig. 7: The JAC molecular model on multiple nodes shows (a) DYAD is 5.3 times faster in producer data movement than Lustre and (b) DYAD is 5.8 times faster in consumer data movement than Lustre and is 192.0 times faster overall.

time when scaling the number of nodes hosting producers and consumers up to 64 (up to 256 producer-consumer pairs) using DYAD and Lustre. Figures 7(a) and 7(b) show the data production time in microseconds for a producer writing and the data consumption time in milliseconds for a consumer reading data to and from either DYAD or Lustre across the various ensemble sizes (8, 16, 32, 64, 128, and 256 producer-consumer pairs) for the JAC MD simulation. The 880-step stride remains the same. In this test, the number of nodes used varies from 2, 4, 8, 16, 32, and 64 nodes; the number of pairs scales with the number of nodes, specifically 8 producers per node (which are 8, 16, 32, 64, 128, and 256 producer-consumer pairs based on the number of nodes involved). Both figures compare the data production time between DYAD and Lustre, with separate bars indicating the time for data movement (red striped bars) and idle time (blue striped bars).

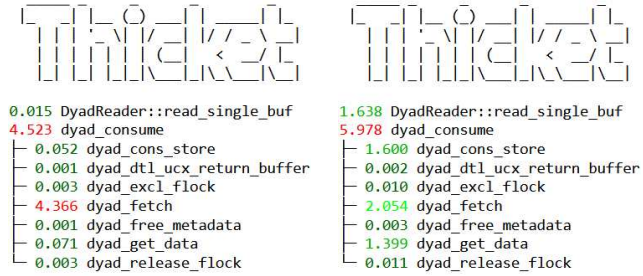
In Figure 7(a), we observe that the data production time for both DYAD and Lustre is stable as the ensemble size increases. The small size of the transferred frame for JAC significantly helps DYAD as it uses node-local storage, but the small size of the frame does not allow for maximal I/O bandwidth from Lustre. Consequently, DYAD is 5.3 times faster than Lustre. For the 128 and 256 ensembles, producers experience less variability in performance across runs by using DYAD. The same configurations exhibit more cost for Lustre, potentially due to interference from other jobs.

In Figure 7(b), with the data consumption time similar to



(a) Producer Performance (b) Consumer Performance

Fig. 8: For scaling molecular models across two nodes a) DYAD scales from 2.1 to 6.3 times faster than Lustre for producer data movement and b) DYAD's consumer data movement scales from 1.6 to 6.0 times faster than Lustre, resulting in an overall 232.0 times faster performance for DYAD.



(a) JAC (b) STMV

Fig. 9: Detailed analysis of DYAD using Thicket shows that as we increase the size of the molecular model, the data movement through `dyad_get_data` and `dyad_cons_store` improve in throughput due to large transfer size. Additionally, the stress on KVS is reduced due to larger data movement, as seen by `dyad_fetch` call, improving the scalability of DYAD.

the small-scale distributed consumption case, data movement for DYAD is 5.8 times faster than for Lustre due to node-local storage. Additionally, the coarse-grained synchronization again adversely impacts the total Lustre consumption time, making DYAD 192.0 times faster than Lustre overall. Additionally, we observe the data movement in both DYAD and Lustre scales as we increase the ensemble size.

To sum up, using node-local storage in DYAD results in substantially faster performance by a factor of 5.3 times for data production and 192.0 times for data consumption. Because of its better performance in data production and consumption times, DYAD's adaptive synchronization mechanisms and use of node-local data management make it a compelling choice for large-scale distributed MD workflows.

Finding 3: Selecting a data management solution that optimizes both data movement and synchronization ensures overall performance in a large-scale, distributed MD workflow.

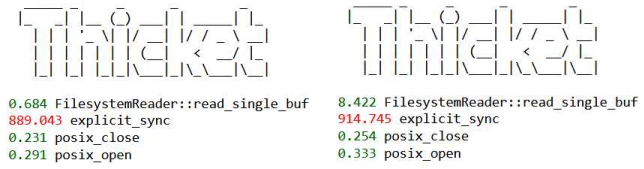
E. Molecular Model Size Scaling

We evaluate molecular model size scaling by comparing the performance of DYAD and Lustre across a range of molecules (i.e., JAC, ApoAI, FI ATPase, and STMV) that increase in size in terms of the number of atoms and consequently in frame size exchanged between producer and consumer.

Figures 8(a) and 8(b) compare the production and consumption times in the context of JAC, ApoA1, F1 ATPase, and STMV with incremental frame sizes (Table I) for DYAD and Lustre. The settings for these tests involve 2 nodes and 16 producer-consumer pairs, with varying strides (granularity or intervals at which data was processed in the simulation) for each molecular model (Table II). These strides are selected so that the MD simulation time for each molecular model is constant. Figure 8(a) presents the data production time in milliseconds in terms of data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre for the four different molecular models. Increasing the size of the molecular model increases the amount of data exchanged between the producer and consumer. Therefore, we see that as we increase size, the data production time increases for both Lustre and DYAD. If we compare DYAD with Lustre, we see that the production cost gap between the two solutions increases; DYAD is 2.1 to 6.3 times faster than Lustre. There is no substantial idle time for either DYAD or Lustre.

Figure 8(b) depicts the data consumption time in milliseconds for the same molecular models as in Figure 8(a). As with the data production times, each model has data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre. The performance of DYAD's data movement improves relative to Lustre with a larger molecular model, increasing from 1.6 to 6.0 times faster than Lustre. This improvement can be attributed to faster node-local storage and efficient RDMA-based data movement. Again, similar to previous results, Lustre's idle time remains higher than that of DYAD, and the overall consumption time in Lustre is dominated by idle time. As a result, DYAD is 121.0 to 333.8 times faster than Lustre.

We use Thicket to analyze the call trees of the MD workflows and pinpoint specific regions that the workflows spend time on. Figure 9(a) and 9(b) show DYAD's performance for the JAC and STMV molecular models. In these trees, not ordered by time, the workflow calls `dyad_consume` which consists of two data movements: (i) move data from a remote node shown with `dyad_get_data` and (ii) store the data into a local cache with `dyad_cons_store`. Additionally, the KVS synchronization is performed using `dyad_fetch`. Finally, the reading of data by the consumer is shown with `read_single_buf`. In this workflow, DYAD handles larger frames more efficiently for data movement. Specifically, based on Table I, we note that we move 45.3 times more data with STMV than with JAC, but the additional cost of data movement is only 33.6 times greater. This indicates that DYAD scales well for larger molecular models. Finally, larger



(a) JAC (b) STMV

Fig. 10: Through detailed analysis of Lustre using Thicket, we observe that Lustre scales well for the data movement with *FilesystemReader::read_single_buf* of larger ensembles due to the inherent parallelization of Lustre file system. However, explicit synchronization with *explicit_sync*, which remains constant across the two runs, reduces the scalability of Lustre.

data movement time reduces the stress on KVS, making the synchronization cost 2.1 times better for STMV than JAC (`dyad_fetch` in Figure 9(a) vs. `dyad_fetch` in 9(b)).

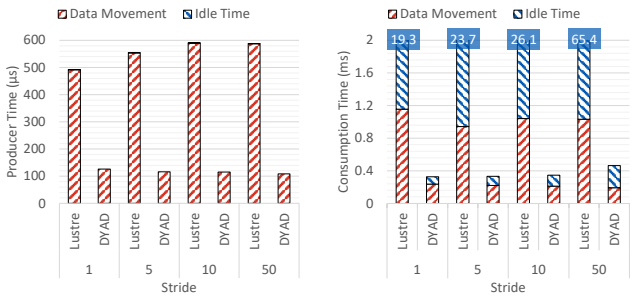
Figures 10(a) and 10(b) show Lustre's performance for the JAC and STMV molecular models. A detailed visualization of the Thicket call tree demonstrates synchronization (`excplicit_sync` in the trees) restricts the scalability for larger molecular models for Lustre. Here, the data movement cost is given by *FilesystemReader::read_single_buf*, and the synchronization cost is given by *explicit_sync*. As we can see, increasing the amount of data by 45.3 times as we go from JAC to STMV results in a 12.3 times increase in data movement time. However, the synchronization cost remains roughly the same, limiting Lustre's overall scalability for MD workflows.

Finding 4: Leveraging local resources and efficient communication protocols enables better scalability as data sizes increase.

F. Frame Generation Frequency Scaling

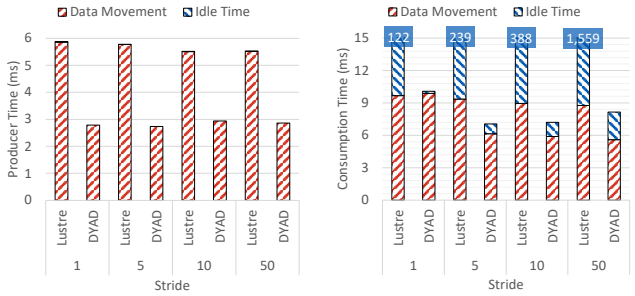
We evaluate frame movement frequency scaling by comparing the performance of DYAD and Lustre across different strides (i.e., generate an MD frame in output every 1, 5, 10, and 50 MD steps), moving data from producers to consumers for both our smallest molecular model (JAC) and our largest model (STMV). For both molecular models, the settings for our tests involve 2 nodes and 16 producer-consumer pairs.

Figures 11(a) and 11(b) compare production and consumption times when using JAC. Figure 11(a) presents the data production time in microseconds in terms of data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre. Across all strides, the data movement time of DYAD and Lustre remains constant, suggesting that both data management solutions can handle a high rate of frame movements. Lustre exhibits some variability in data movement due to possible contention on the Lustre file system. Additionally, we observe that DYAD is 4.8 times faster than Lustre for data production with JAC due to faster node-local storage. Figure 11(b) presents the data consumption time in milliseconds in terms of data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre. We



(a) Producer Performance (b) Consumer Performance

Fig. 11: Tests on two nodes with JAC demonstrate that (a) DYAD's production time is 4.8 times faster than Lustre, and (b) DYAD's consumption time constantly outperforms that of Lustre, both in data movement and idle time.



(a) Producer Performance (b) Consumer Performance

Fig. 12: Tests on two nodes with STMV demonstrates that (a) DYAD production time is 2.0 times faster than Lustre; (b) DYAD consumption time is 13.0 to 192.2 times faster than Lustre due to a widening gap in idle time.

note that the data movement time of DYAD and Lustre remains constant, and DYAD exhibits less variability than Lustre. We also observe that DYAD is 4.8 times faster than Lustre for data movement across the strides. Furthermore, idle times increase with the stride for both DYAD and Lustre. However, DYAD's idle times are much smaller than those of Lustre due to DYAD's adaptive synchronization. As the stride increases, the gap between the overall time of DYAD and Lustre widens. As a result, DYAD becomes even more efficient than Lustre.

Figures 12(a) and 12(b) compare the production and consumption times when using STMV for the same stride range. Figure 12(a) presents the data production time in milliseconds in terms of data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre. We observe that the data movement time of DYAD and Lustre remains constant across all strides, with Lustre showing some variability due to file system contention. Overall, DYAD is 2.0 faster than Lustre due to faster node-local storage. Figure 12(b) presents the data consumption time in milliseconds in terms of data movement (red striped bars) and idle time (blue striped bars) for both DYAD and Lustre. We

observe that Lustre's data movement time is constant across strides. On the other hand, we observe that increasing the stride improves DYAD's data movement performance by up to 1.4 times as compared to a stride of 1. This improvement in performance at higher strides suggests that DYAD experiences lower network contention as stride increases. As a result of the different behaviors of Lustre and DYAD, we observe that DYAD ranges from performing equally as well as Lustre for a stride of 1 to being 1.56 times faster for a stride of 50. DYAD's overall consumption time is 13.0 to 192.2 times faster than Lustre because of DYAD's lower idle times. As with JAC, DYAD's better overall time is due to adaptive synchronization.

Finding 5: Minimizing synchronization is critical as the frequency of data transfer between producers and consumers decreases over a fixed amount of time.

V. Conclusions

This empirical study of data transfer for molecular structures in MD workflows across different data management solutions (i.e., DYAD, XFS, and Lustre) provides insights into optimizing data management for such workflows. This study outlines the impact of fine-grained synchronization and the proper choice of storage setup in optimizing MD workflows' performance and scalability. Despite a slight increase in data production time due to additional metadata management, DYAD's efficient handling of idle times through fine-grained synchronization mechanisms significantly enhances overall workflow efficiency, particularly on single-node configurations. Leveraging network communication for data movement in small-scale distributed MD workflows has minimal effect on performance when the communication is directly between nodes. This suggests that management solutions like DYAD, which utilize direct memory access capabilities for efficient data transfer, can maintain consistent performance even in distributed environments, emphasizing the adaptability of MD workflows to various management solutions. In large-scale distributed MD workflows, selecting a data management solution that optimizes data movement and synchronization processes ensures overall performance and scalability. DYAD's use of node-local storage and adaptive synchronization mechanisms makes it a compelling choice; DYAD significantly outperforms Lustre in data production and consumption times across a wide range of ensemble sizes. As molecular model size increases, leveraging local resources and efficient communication protocols becomes increasingly important. The study demonstrates that DYAD scales better for larger molecular models than Lustre, which is attributed to faster node-local storage and RDMA-based data movement. This scalability is crucial for handling the large data volumes generated in simulations of complex molecular structures. The frequency of frame generation impacts the synchronization and overall performance of the data management solutions. The results suggest that minimizing synchronization overhead is critical, especially as the frequency of data transfer between producers and consumers decreases. DYAD consistently outperforms Lustre in handling higher frequencies of frame generation, highlighting its efficiency in managing synchronization costs and maintaining performance across different strides. The findings of this work improve the understanding of MD workflows, empowering scientists to achieve faster insights in their MD studies. In future work, we will further evaluate DYAD and other management solutions across a more diverse set of workflows, including those which integrate MD simulations.

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 and was supported by the LLNL-LDRD Program under Project No. 24-SI-005 (LLNL-CONF-860358). IL, JM, and MT acknowledge the support of NSF #2138811 and #2331152. The authors acknowledge Jim Garlick and Mark Grondona for their support with Flux and Lauren Whitnah for her feedback on the manuscript.

References

[1] J.-S. 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," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 421–422.

[2] C. Hellwig, "XFS: The Big Storage File System for Linux," *The Magazine of USENIX & SAGE*, vol. 34, no. 5, pp. 10–18, 2009.

[3] P. Schwan *et al.*, "Lustre: Building a File System for 1000-node Clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, 2003, pp. 380–386.

[4] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High Performance Molecular Simulations Through Multi-level Parallelism from Laptops to Supercomputers," *SoftwareX*, 2015.

[5] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Hénin, W. Jiang, R. McGreevy, M. C. R. Melo, B. K. Radak, R. D. Skeel, A. Singharoy, Y. Wang, B. Roux, A. Aksimentiev, Z. Luthey-Schulten, L. V. Kalé, K. Schulten, C. Chipot, and E. Tajkhorshid, "Scalable Molecular Dynamics on CPU and GPU Architectures with NAMD," *The Journal of Chemical Physics*, 2020.

[6] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a Flexible Simulation Tool for Particle-based Materials Modeling at the Atomic, Meso, and Continuum Scales," *Computer Physics Communications*, 2022.

[7] D. A. Case, T. E. Cheatham III, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, "The Amber Biomolecular Simulation Programs," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1668–1688, 2005.

[8] J. A. Dodds, "Satellite Tobacco Mosaic Virus," *Annual Review of Phytopathology*, vol. 36, no. 1, pp. 295–310, 1998.

[9] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl, "Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS," in *Solving Software Challenges for Exascale: International Conference on Exascale Applications and Software, EASC 2014, Stockholm, Sweden, April 2-3, 2014, Revised Selected Papers 2*. Springer, 2015, pp. 3–27.

[10] S. Caino-Lores, M. Cuendet, J. Marquez, E. Kots, T. Estrada, E. Deelman, H. Weinstein, and M. Taufer, "Runtime Steering of Molecular Dynamics Simulations Through In Situ Analysis and Annotation of Collective Variables," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2023, pp. 1–11.

[11] T. M. A. Do, L. Pottier, R. F. da Silva, F. Suter, S. Caíno-Lores, M. Taufer, and E. Deelman, "Co-scheduling Ensembles of In Situ Workflows," in *2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2022, pp. 43–51.

[12] M. Taufer *et al.*, "Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-generation supercomputers," in *2019 15th International Conference on eScience (eScience)*. IEEE, 2019, pp. 188–198.

[13] K.-L. Ma, "In Situ Visualization at Extreme Scale: Challenges and Opportunities," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, pp. 14–19, 2009.

[14] S. Doerr and G. De Fabritiis, "On-the-fly Learning and Sampling of Ligand Binding by High-throughput Molecular Simulations," *Journal of Chemical Theory and Computation*, vol. 10, no. 5, pp. 2064–2069, 2014.

[15] G. A. Tribello, M. Bonomi, D. Branduardi, C. Camilloni, and G. Bussi, "PLUMED 2: New feathers for an old bird," *Computer Physics Communications*, 2014.

[16] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The Evolution of the Pegasus Workflow Management Software," *Computing in Science & Engineering*, 2019.

[17] B. J. Cochran, K.-L. Ong, B. Manandhar, and K.-A. Rye, "APOA1: a Protein with Multiple Therapeutic Functions," *Current Atherosclerosis Reports*, vol. 23, pp. 1–10, 2021.

[18] M. Dittrich, S. Hayashi, and K. Schulten, "On the Mechanism of ATP Hydrolysis in F1-ATPase," *Biophysical Journal*, vol. 85, no. 4, pp. 2253–2266, 2003.

[19] NAMD Performance. [Online]. Available: https://www.ks.uiuc.edu/Research/namd/benchmarks/

[20] NAMD Utilities. [Online]. Available: https://www.ks.uiuc.edu/Research/namd/utilities/

[21] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: Performance Introspection for HPC Software Stacks," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016.

[22] S. Brink, M. McKinsey, D. Boehme, C. Scully-Allison, I. Lumsden, D. Hawkins, T. Burgess, V. Lama, J. Lüttgau, K. E. Isaacs, M. Taufer, and O. Pearce, "Thicket: Seeing the Performance Experiment Forest for the Individual Run Trees," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 2023.

[23] I. Lumsden, J. Luettgau, V. Lama, C. Scully-Allison, S. Brink, K. E. Isaacs, O. Pearce, and M. Taufer, "Enabling Call Path Querying in Hatchet to Identify Performance Bottlenecks in Scientific Applications," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022.

[24] Corona. [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/corona