MFNetSim: A Multi-Fidelity Network Simulation Framework for Multi-Traffic Modeling of Dragonfly Systems

XIN WANG, Computer Science, University of Illinois Chicago, Chicago, United States
KEVIN A. BROWN, Argonne National Laboratory, Lemont, United States
ROBERT B. ROSS, Argonne National Laboratory, Lemont, United States
CHRISTOPHER D. CAROTHERS, Computer Science, Rensselaer Polytechnic Institute, Troy, United States
ZHILING LAN, University of Illinois Chicago, Chicago, United States

In high-performance computing (HPC), modern supercomputers typically provide exclusive computing resources to user applications. Nevertheless, the interconnect network is a shared resource for both inter-node communication and across-node I/O access, among co-running workloads, leading to inevitable network interference. In this study, we develop MFNetSim, a multi-fidelity modeling framework that enables simulation of multi-traffic simultaneously over the interconnect network, including inter-process communication and I/O traffic. By combining different levels of abstraction, MFNetSim can efficiently co-model the communication and I/O traffic occurring on HPC systems equipped with flash-based storage. We conduct simulation studies of hybrid workloads composed of traditional HPC applications and emerging ML applications on a 1,056-node Dragonfly system with various configurations. Our analysis provides various observations regarding how network interference affects communication and I/O traffic.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Discrete-event simulation**; • **Networks** → **Network simulations**; **Network performance analysis**; **Network performance modeling**.

Additional Key Words and Phrases: Multi-fidelity modeling, Network simulation, Performance analysis, Dragonfly interconnect network

1 Introduction

In high-performance computing (HPC), there has been a surge in hardware complexity and diverse workloads. Supercomputers are progressively embracing heterogeneous architectures that integrate various processors and storage configurations. Concurrently, HPC workloads are becoming more diverse, exhibiting multiplex data movement patterns. Typically, the sequential transfer of large files, common in traditional modeling and simulation, is reasonably well-served by cost-effective hard disk drives. However, the I/O handling for artificial intelligence (AI) and machine learning (ML) applications, especially during training, often involves moving a large number of small, randomly accessed files, which is more efficiently managed by higher-cost flash memory.

Modern HPC systems typically provide exclusive computing resources to user applications. However, the interconnect network is a shared resource among co-running workloads. Previously, interconnect networks primarily served the inter-process communication needs of parallel applications. With advancements in storage techniques and the increasing adoption of flash-based storage, interconnect networks are now required to

Authors' Contact Information: Xin Wang, Computer Science, University of Illinois Chicago, Chicago, Illinois, United States; e-mail: xwang823@uic.edu; Kevin A. Brown, Argonne National Laboratory, Lemont, Illinois, United States; e-mail: kabrown@anl.gov; Robert B. Ross, Argonne National Laboratory, Lemont, Illinois, United States; e-mail: rross@mcs.anl.gov; Christopher D. Carothers, Computer Science, Rensselaer Polytechnic Institute, Troy, New York, United States; e-mail: chrisc@cs.rpi.edu; Zhiling Lan, University of Illinois Chicago, Chicago, Illinois, United States; e-mail: zlan@uic.edu.

ACM 1558-1195/2025/4-ART
https://doi.org/10.1145/3729424

serve dual purposes for both *inter-node communication* and *across-node I/O access*. For example, the exascale supercomputer Aurora [1] employs Distributed Asynchronous Object Storage (DAOS), where storage and I/O nodes share the same interconnect network with compute nodes. While the network connection between I/O groups is allocated with larger bandwidth, the concurrent traffic from both communication and I/O access remains a potential bottleneck [3]. The increasing role of interconnect networks introduces complexities and challenges in balancing the competing demands of both types of traffic.

While the inevitability of *network interference* is clear, its comprehension remains limited, particularly in the context of multiplex traffic sharing network links for both communication and I/O access. Existing studies have predominantly focused on analyzing communication interference on the interconnect network. While valuable, a significant gap exists in the research on the joint analysis of communication and I/O interference on interconnect networks. *How to effectively model network interference among multiplex traffic from diverse workloads remains an open problem.*

Our study aims to address this critical gap by providing *a multi-fidelity modeling framework* that enables the simulation of multiple traffic types simultaneously over the interconnect network. By combining different levels of abstraction, MFNetSim, our multi-fidelity framework, is capable of effectively co-modeling the communication and I/O traffic on supercomputers equipped with flash-based storage.

In addition, we conduct simulation studies of hybrid workloads comprised of traditional HPC applications and emerging AI/ML applications on a 1,056-node Dragonfly system equipped with various configurations. The quantitative "what-if" exploration reveals several key findings:

- Both I/O traffic and communication traffic are greatly impacted by network sharing, as they compete for network bandwidth and resources.
- Applications with extensive I/O or intensive communication are more resilient to network interference, while those with less intensive I/O or communication suffer more slowdown in performance.
- Job placement significantly impacts I/O and communication performance. An intelligent and flexible job placement methodology is preferred to strike a balance between commonly used random placement and contiguous placement.

The remainder of this paper is organized as follows: Section 2 introduces related work. Section 3 to 5 present the proposed system architectures, the multi-fidelity design, and its implementation. Section 6 provides model validation. Section 7 describes case studies utilizing our framework. Section 8 presents the experimental results and analysis, followed by Section 9 summarizing our conclusions.

2 Background and Related Work

2.1 Background

We give an overview of three related software packages, namely CODES, Union, and coNCePTuaL, which serve as building blocks for our proposed multi-fidelity modeling framework.

CODES (Enabling CO-Design of Exascale Storage Systems) is a parallel, event-driven simulator designed for high-fidelity, packet-level simulations, enabling the exploration of large-scale network and storage architectures [12]. Built on the Rensselaer Optimistic Simulation System (ROSS), a discrete event simulation framework, CODES leverages parallel execution to enhance scalability. One of its core features is network simulation, which includes a modular abstraction layer that supports a variety of network topology models such as dragonfly, Torus, Fat-Tree, Slim Fly, and others [11, 22, 24]. Additionally, CODES features a workload generator that abstracts I/O and network workloads from multiple sources to drive its network and storage models. These sources range from synthetic workloads to application traces and SWM skeletons [13]. CODES provides a local storage model based on the analytical techniques of Ruemmler and Wilkes [19]. The model represents I/O access by scheduling read and write I/O requests in a first-come first-served manner where the available storage space is configurable.

Union provides in-situ workload generation for CODES [21]. Users only need to describe their applications using simple English instructions in coNCePTUaL (described below). Union automatically translates these instructions into a skeleton and coordinates the skeleton generation in CODES. Union contains two main components: a translator that automatically translates coNCePTuaL applications into skeletons, and an event generator that emits communication events from skeletons to CODES as in-situ workloads. Union supports concurrent workloads and also provides a flexible rank-to-node mapping.

The coNCePTuaL (Network Correctness and Performance Testing Language) [16] is a domain-specific programming language aimed at testing and analyzing network performance. It offers a platform-independent approach to defining communication patterns and network evaluations, incorporating constructs to specify message sizes, communication structures, and other key aspects of network performance testing. coNCePTuaL consists of two main components: a domain-specific language (DSL) and a compiler. The DSL is designed to write network benchmarks, featuring a syntax enriched with keywords that encapsulate various built-in functions to simplify the implementation of complex communication patterns. The compiler converts the coNCePTuaL code into low-level instructions, integrating calls to a messaging library [15]. One of its notable capabilities is the inclusion of built-in functions that support various virtual topologies for application communication, such as n-ary trees, meshes, tori, and k-nomial trees. These functions significantly reduce the manual effort required to model complex communication behaviors.

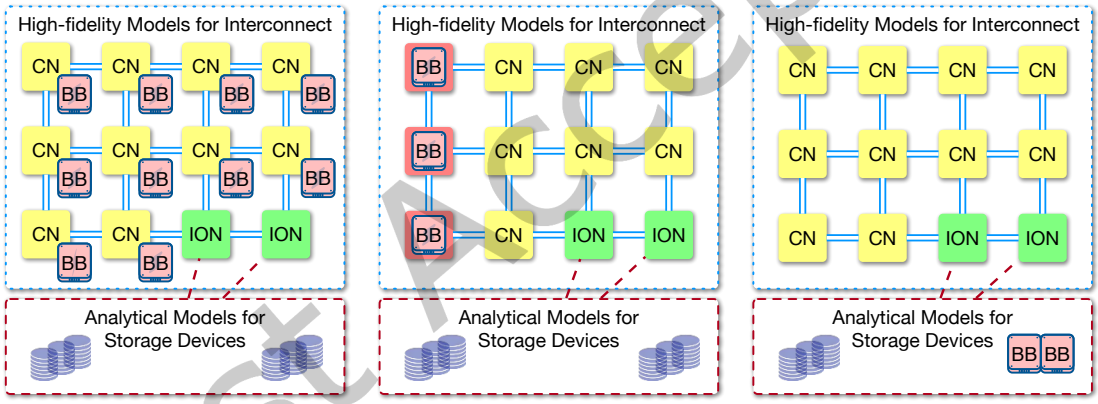


Fig. 1. Overview of system architectures. The computing system serves multiple applications, utilizing the interconnect network for both inter-process communication and I/O data movement. High-fidelity simulation is employed for network packets (blue dashed box), while low-fidelity analytical models are used for storage device operations (red dashed box). CN represents compute node, BB represents burst buffer, and ION represents I/O node.

2.2 Related Work

Since the dragonfly topology has been proposed and adopted by production systems, many simulation studies have been conducted to explore network interference in a multi-application environment on dragonfly. Existing simulators developed for high-performance computing architectures include Booksim [4], SST/macro [18], CODES [12], and others.

Previous studies mainly focused on *the communication performance* of applications. Studies explore inter-application interference using a trace-based simulation framework. Quantitative analysis for multi-application communication interference has been conducted using trace replay [22, 23], synthetic patterns [6], or skeleton in-situ workload simulation [21]. At the routing level, Kang et al. have explored reinforcement learning routing for workload interference migration on Dragonfly [5]. These studies show that communication-intensive applications affect less intensive applications. Furthermore, communication pattern, along with job placement and routing mechanism, greatly affects the communication performance.

Methodologies for co-analysis of MPI communication and I/O are also proposed in several studies. Kunkel proposed a sequential discrete event simulator PIOsimHD which enables simulation of collective communication patterns as well as simulation of parallel I/O with analytical models [8]. Mubarak et al. have analyzed the effects of interference of checkpointing I/O and uniform random network traffic on burst-buffer-equipped dragonfly-based systems using a high-resolution simulation [11]. They conducted experiments with different routing strategies to show that balancing I/O and network traffic requires a careful selection of routing mechanisms, and job and data placement.

Our work differs from these studies in several aspects. First, while existing studies focus on isolated communication or I/O interference, MFNetSim enables *joint analysis of MPI and I/O traffic* by simulating hybrid workloads with distinct communication and I/O patterns. Second, instead of relying on single-application traces or synthetic patterns, our framework supports multi-traffic skeletons (Section 7.2), allowing us to model interference between diverse workloads such as HPC, ML, and checkpointing applications. This capability provides unique insights into how multiplex traffic competes for shared network resources, a critical gap in existing tools like PIOsimHD or trace-based simulators. We believe our *multi-traffic interference study* offer valuable insights to the HPC community regarding workload interference among multiplex communication and I/O operations generated from diverse applications.

3 System Architecture

I/O nodes and compute nodes in production systems are linked via an interconnect network. Application processes are assigned to specific compute nodes according to the configuration files. The assigned compute nodes run and issue MPI and I/O requests.

We model the system architectures with three different burst buffer configurations: local burst buffer, compute-side global burst buffer and storage-side burst buffer, as shown in Figure 1. Here, CN stands for compute node, BB stands for burst buffer, and ION stands for I/O server node. It is important to note that high-fidelity simulation is employed for network packets (blue dashed box), while low-fidelity analytical models are used for storage device operations (red dashed box).

In the local burst buffer setting, each compute node pocesses a local burst buffer, serving as a node-local I/O cache. Data transfer between compute nodes and the local burst buffer does not interfere with traffic on the interconnect network.

In the compute-side global burst buffer setting, burst buffer devices occupy a subset of nodes in the interconnect network, ensuring that one router in each group is directly connected to burst buffer nodes. Data exchange between compute nodes and burst buffer devices utilizes the bandwidth of the shared network in this configuration. Burst buffer devices communicate with external HDD storage through I/O server nodes.

In the storage-side burst buffer setting, the external storage comprises hybrid SSD and HDD devices, with SSD devices utilized for small-size I/O (less than 64KB) and HDD devices for large-size I/O. In this scenario, the compute nodes communicate with the I/O server nodes for I/O operations.

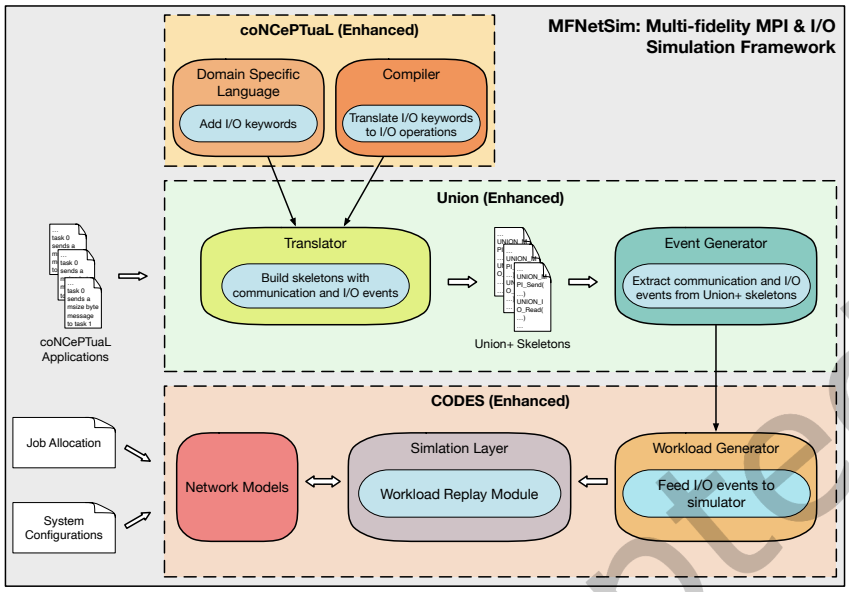


Fig. 2. Key components of MFNetSim. It is built on three packages: coNCePTuaL, Union, and CODES. Our enhancements are shaded in blue.

4 Multi-fidelity Modeling

To enable scalable simulation of multiplex traffic, we develop MFNetSim, a multi-fidelity framework that integrates *high-fidelity packet-level network simulation* for MPI and I/O traffic with *low-fidelity analytical storage models* for I/O access. This partitioning prioritizes network fidelity to accurately model network behavior, while simplifying storage modeling to avoid prohibitive computational costs. This design choice is driven by the need to efficiently study network interference. Modeling both communication and I/O at high fidelity would be computationally impractical for large-scale studies, whereas an entirely low-fidelity approach would fail to capture critical network dynamics. By prioritizing network fidelity while simplifying storage modeling, MFNetSim achieves a practical balance between simulation accuracy and performance.

As shown in Figure 2, our design accommodates both MPI and I/O operations accordingly. The data movements in the interconnect network are simulated in packet-level discrete-event simulation, while the data movements in the storage area network are calculated using analytical models, thus providing *multi-fidelity for multi-traffic modeling*. By prioritizing network performance analysis, MFNetSim substitutes I/O operations on storage devices with delay models. This substitution significantly reduces simulation costs without compromising accuracy. Communication and I/O operations across the interconnect network, which may lead to interference, are simulated with packet-level detail.

4.1 High-fidelity Network Modeling

Parallel discrete-event simulations are made up of logical processes (LPs), where each LP models a component of the system and has a distinct state. These LPs interact with one another via events in the form of timestamped messages. In order to accurately simulate the MPI operations coming from scientific applications, the MPI simulation layer takes the operations from the CODES workload generator and simulates the MPI tasks on top of the network models while maintaining the correct causality order. We extend this MPI communication model to

include simulating both MPI and I/O operations from workload generator, creating the workload replay module. The workload replay module consists of separate models for MPI and I/O operations, namely the MPI handler and the storage manager. The MPI handler imports the MPI replay model from CODES, while the storage manager enhances the storage server model [11] to adopt three different system architectures discussed in Section 3.

The parallel file systems in HPC systems often employ dedicated I/O servers or I/O nodes responsible for managing and coordinating I/O operations. Compute nodes communicate with these dedicated I/O nodes to initiate file read and write operations. The I/O nodes act as intermediaries between the clients and the underlying storage devices. I/O nodes are responsible for coordinating I/O requests and managing concurrent access to files from compute nodes. Meanwhile, to meet the new requirements in the exascale computing era, a flash-based storage tier between the compute cluster and a slower disk-based storage tier has been introduced to provide a faster storage resource to the supercomputer. To model the data access between compute nodes, I/O nodes and storage devices, the storage manager model is composed of four LPs: a compute node LP, a storage manager LP, a HDD LP and a SSD LP. The storage manager LP manages I/O to/from HDD LP and SSD LP.

Figure 3, 4, and 5 illustrate the interaction between LPs for three different burst buffer setups. Black arrows represent the simulated I/O events that do not generate network traffic, blue arrows show events causing traffic on the interconnect network, and red arrows indicate events between the compute network and storage area network. Delays for the black and red arrow events are calculated using analytical models, while delays for the blue arrow events are tracked in the discrete-event simulator.

Local burst buffer. In this setup, the local burst buffer can be treated as a cache. Any I/O operations initiated at compute nodes will first be directed to the SSD LP. In this configuration, the SSD LP will make decisions about when to contact the storage manager. Figure 3 shows the interaction between the LPs.

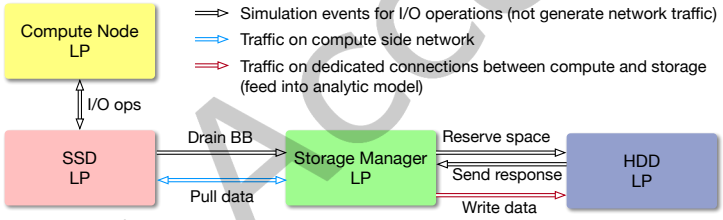


Fig. 3. Interaction between LPs for Local BB configuration

Compute-side global burst buffer. Upon receiving write requests from compute node LPs, the storage manager LP initiates a blocking call to the SSD LP in order to reserve the requested space. Once space is reserved on the SSD LP, the storage manager begins a series of concurrent, pipelined RDMA reads of fixed-sized blocks from the requester and data writes to the SSD LP. Figure 4 illustrates the interaction between the LPs.

Storage-side burst buffer. Similar to the compute-side global burst buffer setup, the storage manager LP receives write requests from compute node LPs and makes a blocking call to either the SSD LP or the HDD LP based on a configurable message size threshold. Figure 5 shows the interaction between the LPs.

4.2 Analytical Storage Modeling

For HDD modeling, we utilize the analytical model introduced by Parsons et al. [17]. This model does not require time-consuming and error-prone physical parameter extraction from a real disk. Instead, it relies on a few parameters derived from typical disk characteristics, which can be manually identified.

When there are no requests in the queue, the request latency is simply the summation of the seek time T_{seek}, rotational latency γ, and the transfer time $T_{transfer}$, as shown in Equation 1. Transfer time is calculated using the

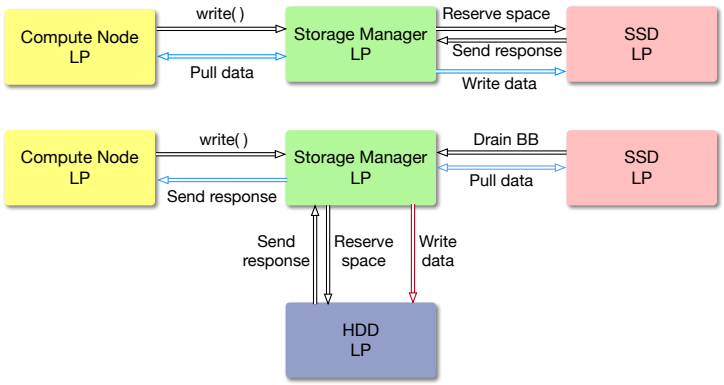


Fig. 4. Interaction between LPs for Compute-side BB configuration

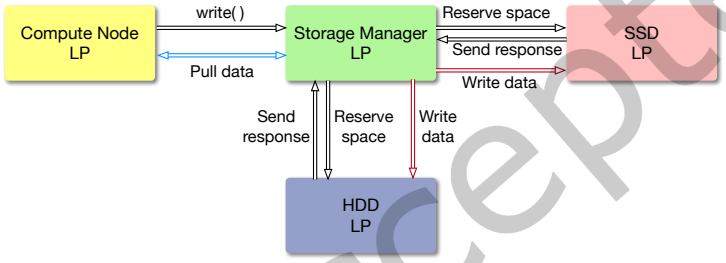


Fig. 5. Interaction between LPs for Storage-side BB configuration

length of the tracks in the current zone (L_{track}), the length of the request ($L_{request}$), and the disk rotational speed ($diskRPM$). T_{seek} and L_{track} are defined with a set of constant parameters which can be calculated using square-root/linear regression. Rotational latency is modeled as a random time between the minimum and maximum latency.

$$\begin{aligned} Latency &= T_{seek} + \gamma + T_{transfer} \\ T_{transfer} &= \frac{L_{request} \times L_{track} \times 60}{diskRPM} \end{aligned} \tag{1}$$

If a request is received while a prior request is being handled, it is placed in the queue until the current request completes. After each request, the disk will perform at least the minimum readahead, which is parameterized with a proper threshold. If a request can be partially fulfilled with readahead data, the request latency is only composed of the remaining transfer time for the unread data. When a request can be fulfilled with data from the cache, that request is immediately returned. Most of the parameter values can be identified from the disk specification, while the others can be determined through simple benchmarks.

For SSD modeling, we employ the analytical model proposed by Yoo et al. [25], which can precisely compute the latency of a read (or write) request. This model facilitates the modeling of parallel behavior in SSDs using a single thread. For example, upon receiving an I/O request, the thread calculates the I/O latency with the latency model and applies an appropriate amount of delay using the busy waiting method. With the proper setting of parameters, this method can accurately calculate I/O delays in multi-channel/multi-way SSDs.

In this model, the write and read latency are represented by Equation 2. For example, the latency of write is a function of channel switching delay(W_{ch}), single page write latency (W_{page}), number of pages (N_{page}), number of cycles (N_{cycle}) and wait time (T_{wait}). The wait time is defined in Equation 3, where ρ is the maximum number of I/Os per cycle.

$$\begin{aligned} T_w &= W_{ch} \times (N_{page} - 1) + T_{wait} \times (N_{cycle} - 1) + W_{page} \\ T_r &= R_{ch} \times (N_{page} - 1) + R_{page} \end{aligned} \tag{2}$$

$$T_{wait} = \begin{cases} W_{page} - W_{ch} \times \rho & \text{if } W_{page} > W_{ch} \times \rho \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

All parameters in the above equations can be obtained through device specifications and benchmarking. By carefully selecting appropriate parameters, we integrate the HDD and SSD models into our multi-fidelity framework for accelerated workload simulation, while achieving satisfactory accuracy.

5 Implementation

In constructing an integrated framework for multi-traffic simulation, we have developed the following enhancements in coNCePTuaL, Union, and CODES to enable simultaneous simulations of I/O and MPI.

Enhancement of coNCePTuaL. We enhance coNCePTuaL to include the capability to translate and compile I/O operations. Specifically, we introduce keywords such as READ, WRITE, OPEN, and CLOSE to the domain-specific language, allowing users to incorporate I/O operations into coNCePTuaL programs. Figure 6 illustrates an example coNCePTuaL program with the newly added I/O keywords highlighted at line 16. Additionally, we update the C+MPI compiler to translate instructions containing these new keywords into appropriate I/O operations in the C language.

```
1  Require language version "1.5".
2
3  # Parse the command line.
4  numwords is "Message size (words)" and comes from
   "--msgsize" or "-s" with default 14413K.
5  reps is "Number of repetitions" and comes from "--reps" or
   "-r" with default 100.
6  sample is "Sample size" and comes from "--sample" or "-f"
   with default 8M.
7  computetime is "Computation time (ms)" and comes from
   "--compute" or "-c" with default 129.
8
9  # Allocate a send buffer and a receive buffer.
10 Task 0 multicasts a numwords*num_tasks word message from
   buffer 0 to all other tasks.
11 Task 0 multicasts a numwords*num_tasks word message from
   buffer 1 to all other tasks.
12
13 # Measure the performance of CODES_MPI_Allreduce().
14 Task 0 resets its counters then
15 for reps repetitions {
16     all tasks READ sample then
17     all tasks COMPUTES FOR computetime MILLISECONDS then
18     all tasks backend execute "
19         UNION_MPI_Allreduce([MESSAGE BUFFER 0], [MESSAGE
           BUFFER 1], (int)" and numwords and ",
20                 MPI_INT, MPI_SUM, MPI_COMM_WORLD);
21         "
22 } then
23 task 0 logs elapsed_usecs/1000 as "Elapse time (ms)".
```

Fig. 6. An example of an enhanced coNCePTuaL program supporting both I/O and MPI communication keywords. The newly added I/O keyword is highlighted with a blue frame.

```
1  /* Process a subset of the events in a given event list. */
2  static void conc_process_events (CONC_EVENT *eventlist,
3  ncptl_int firstev, ncptl_int lastev, ncptl_int numreps)
4  {
5    ...
6    case EV_READ:
7      (void) UNION_IO_READ((int)thisev->s.io.fid, NULL,
8          thisev->s.io.size);
9    case EV_CODE:
10     /* Execute an arbitrary piece of code. */
11     switch (thisev->s.code.number) {
12     case 0:
13       UNION_MPI_Allreduce((ncptl_get_message_buffer((
           ncptl_int)(0))), (ncptl_get_message_buffer((
           ncptl_int)(1))), thisev->s.code.var_numwords,
           MPI_INT, MPI_SUM, MPI_COMM_WORLD);
14     break;
15   break;
16   ...
17 }
18
19 /* Program execution starts here. */
20 static int cosmoflow_main (int argc, char *argv[])
21 {
22 CONC_EVENT * eventlist; /* List of events to execute */
23 ncptl_int numevents; /* Number of entries in eventlist[] */
24 ...
25 conc_initialize (argc, argv);
26 conc_process_events (eventlist, 0, numevents-1, 1);
27 return conc_finalize();
28 }
29
30 /* fill in function pointers for this method */
31 struct union_conceptual_bench cosmoflow_bench =
32 {
33 .program_name = "cosmoflow",
34 .conceptual_main = cosmoflow_main,
35 };
```

Fig. 7. A code snippet of a Union skeleton application with enhanced I/O operation handler. Line 6-8 intercept and translate the I/O operation (line 16 of Figure 6) to Union I/O interfaces. Some portions of the code have been omitted for brevity.

Enhancement of Union. We add a set of Union I/O interfaces to accommodate both MPI and I/O operations from workloads and process them accordingly. I/O event handlers are integrated into the Union translator to intercept I/O operations. The enhanced translator converts all communication and I/O function calls to utilize the Union interfaces. For example, the `READ()` call is transformed into `READ()`, as illustrated in Figure 7 (lines 6-8).

Additionally, we incorporate the declaration of I/O operations into the event generator and implement corresponding functionality in the CODES workload generator. This enables both MPI and I/O operations from Union skeletons to be emitted as simulation events in CODES. Figure 7 presents a code snippet of a Union skeleton with enhanced I/O operations, where an example Union I/O interface is shown in lines 6-8.

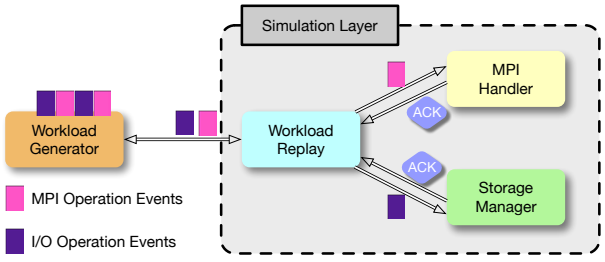


Fig. 8. An illustration of workload replay module.

Enhancement of CODES. The enhancement of CODES primarily focuses on the network module. We develop a separate package called the workload replay module, capable of handling both MPI and I/O operation events from the workload generator. MPI events are directed to the MPI event handler for processing, while I/O events are forwarded to a storage server model, as depicted in Figure 8.

The MPI handler imports the MPI replay module from CODES to accurately simulate MPI operations. The storage manager utilizes our high-fidelity network modeling for I/O operations, as discussed in Section 4.1. Additionally, analytical models for HDD and SSD are integrated into the local storage model, and the burst buffer storage server model is enhanced to accommodate three burst buffer setups.

6 Storage Model Validation

While CODES has been previously validated for communication performance with the Theta system [14], this work focuses on validating the analytical models for HDD and SSD to ensure accurate storage performance simulation. The integration of Union, CODES, and coNCePTuaL was validated in prior work [21]. To further assess the accuracy of our storage models, we compared the theoretical performance predicted by the latency models with the observed performance on two physical storage devices: the Intel D3-4510 SSD and the Seagate ST2000NX0253 HDD. These comparisons were conducted following similar methodologies as outlined in [25], utilizing baremetal hosts available on Chameleon [7].

We compared the latency predictions generated by the single-page write/read latency model with the actual I/O performance exhibited by the HDD and SSD devices across four workloads: sequential write/read and random write/read. For sequential I/O workloads, we sequentially wrote and read a 512 MB file with a 512 KB record size. Conversely, for random I/O workloads, we performed write and read operations on a 512 MB file at random offsets with a 4 KB page size. The performance results obtained from both the physical devices and our storage models for these four workloads are summarized in Table 1. The error rates of the latency model were found to be less than 5% for SSD and less than 10.3% for HDD, respectively.

The 10.3% error observed in HDD modeling is acceptable given our focus on network interference analysis. While a higher-fidelity storage model could reduce this error, it would significantly increase simulation runtime without enhancing insights into network contention. Our goal is to capture the relative impact of co-running traffic on network links, not absolute storage performance. This trade-off aligns with our design philosophy: prioritizing high-fidelity network simulation for accurate interference studies while using analytical storage models to ensure scalability.

Our validation ensures that the storage models provide sufficient accuracy within the multi-fidelity framework. Since our primary objective is to study network interference, not full-system behavior, end-to-end validation is beyond the scope of this work. While such validation could provide additional insights, it would require extensive real-world job traces and system measurements, which are not the focus here. Instead, our evaluation demonstrates that the storage models achieve the necessary accuracy for analyzing interference effects in large-scale HPC systems.

7 Case Study

In this section, we present a case study that utilizes MFNetSim for multi-traffic workload interference analysis. We perform simulations of hybrid HPC and ML workloads on a 1,056-node Dragonfly system, exploring different placement and routing mechanisms.

7.1 Dragonfly System

We simulate a 1,056-node Dragonfly system as specified in Table 2. The network bandwidth is configured according to a Cray Cascade system, Theta [2], with a global link bandwidth of 4.37 GiB/s, local link bandwidth between routers of 5.25 GiB/s, and channel bandwidth between routers and compute nodes of 16 GiB/s. For this

Table 1. Validation of Write/Read Latency Models

Workload		Model	Disk	Error
HDD	Seq Read	136.00 MB/s	123.33 MB/s	10.3%
	Seq Write	136.00 MB/s	129.00 MB/s	5.4%
	Rand Read	111.69 IOPS	122.74 IOPS	9.0%
	Rand Write	592.22 IOPS	577.91 IOPS	2.5%
SSD	Seq Read	392.04 MB/s	408.69 MB/s	4.6%
	Seq Write	388.26 MB/s	378.96 MB/s	5.0%
	Rand Read	8474.58 IOPS	8906.43 IOPS	4.8%
	Rand Write	14084.51 IOPS	13772.85 IOPS	2.3%

Table 2. System Configuration

Topology	Radix	#Groups	#Routers/ Group	#Nodes/ Router	#Global links/ Router	#Links between Groups	System Size
1D Dragonfly	16	33	8	4	4	1	1056

Table 3. Hybrid Workloads and Applications

Application	Communication Pattern	I/O pattern	In Workload 1	In Workload 2
MILC	Nearest neighbour & many-to-many	Reading configurations	Yes	No
Nekbone	Nearest neighbour & many-to-many	Reading configurations	No	Yes
Checkpoint	Synchronization	Writing state periodically	Yes	Yes
Cosmoflow	Allreduce	Reading batches every step	Yes	Yes

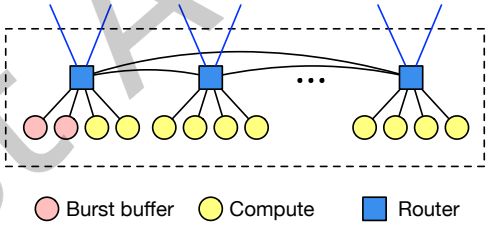


Fig. 9. Dragonfly group configuration. The first two nodes within a group are burst buffer nodes, the remaining 30 nodes are compute nodes.

case study, we adopt the compute-side global burst buffer setting. Each group consists of 32 nodes, with the first two nodes serving as burst buffer nodes and the remaining 30 nodes as compute nodes, resulting in a total of 990 compute nodes and 66 burst buffer nodes. Figure 9 illustrates the Dragonfly group configuration for this case study. Following recommendations from [11], the I/O nodes route the I/O traffic to the nearest burst buffer nodes.

We adopt the compute-side global burst buffer configuration for this case study because it inherently amplifies cross-application interference. Unlike the local burst buffer, where I/O traffic avoids the shared network, or the storage-side configuration, where I/O traffic may prioritize dedicated paths, the compute-side setup forces both MPI and I/O traffic to compete directly for shared network bandwidth. This configuration creates a contention

hotspot between compute nodes and burst buffer nodes, mirroring real-world scenarios in systems like Aurora [1], where I/O nodes are integrated into the interconnect network, making it a relevant scenario for our study.

7.2 Hybrid Workloads

We investigate two hybrid workloads composed of a mix of HPC and ML applications. Table 3 shows the organization of two hybrid workloads and the characteristics of applications. The applications are chosen to cover various representative communication and I/O traffic patterns in our experiments. The studied applications can be categorized into three groups based on their communication and I/O characteristics: (i) MILC and Nekbone are typical HPC applications with distinct communication intensities and negligible I/O; (ii) Cosmosflow represents typical communication and I/O patterns observed in machine learning applications; (iii) Checkpoint represents a widely used fault tolerance application in HPC, featuring high-intensity bursts of I/O. The hybrid workloads are composed by selecting one application from each group. The details of each application are listed below:

MILC. This application is developed by the MIMD Lattice Computation (MILC) collaboration to study quantum chromodynamics (QCD) computations on parallel computers. It is commonly used for simulating the behavior of quarks and gluons within a lattice framework. It performs simulations of four dimensional SU(3) lattice gauge theory. MILC is configured with 300 ranks, each rank issues nonblocking send and receive messages of size 486 KB to communicate with neighbors. It has negligible I/O operations for reading configurations.

Nekbone. Nekbone is a mini-app derived from the computational fluid dynamics code Nek5000. Nekbone solves a standard Poisson equation using a conjugate gradient iteration with a simple preconditioner. Nekbone is configured with 300 ranks, performing a large number of MPI collective operations with small 8-byte messages. It uses nonblocking send and receive to transmit messages with various sizes from 8 bytes to 165 KB. The only I/O operation is reading a small configuration file (336 bytes) during setup, which is negligible.

Checkpoint. The application periodically saves its state to stable storage to allow for resuming execution later in case of interruptions. Specifically, the application writes to a file at specified time intervals. It is configured with 300 ranks and writes a total of 50GB of data to a file. The application processes must coordinate and communicate to ensure a consistent state is captured.

Cosmoflow. Distributed machine learning algorithms are featured with periodic Allreduce calls to gather gradients from multiple worker nodes and broadcast summation result to them. This application captures this feature by iteratively issuing Allreduce calls with a predefined compute time interval. For I/O, the I/O threads read training samples at each step time, which is pipelined with gradient computation. It is configured as a 300-rank job that issues 28.15 MB Allreduce messages and 8 MB read request every 129 ms as described in [10].

7.3 Job Placement and Routing

We investigate two commonly used job placement policies and two commonly used routing algorithms as described below.

- **Random Nodes (rand)** placement selects compute nodes for each job completely randomly from the entire system. Compute nodes that connect to the same router tend to be assigned to different jobs.
- **Contiguous (cont)** placement assigns each job available nodes consecutively. This method tends to place application processes into the same group.
- **Minimal Routing (min)** routes a packet along the minimal path from source to destination. Minimal routing can guarantee the minimum hops a packet traverses.
- **Progressive Adaptive Routing (PAR)** selects packet paths based on congestion situations on minimal and non-minimal paths. If a non-minimal path is chosen, the packet is routed minimally to a random intermediate router before being forwarded minimally to its destination. PAR permits source group routers to re-evaluate routing decisions during minimal routing.

7.4 Performance Metrics

The performance metrics we analyze include *communication time*, *I/O time*, *message latency*, *link traffic*, and *link saturation time*. Communication time refers to the portion of process runtime used for sending and receiving messages. I/O time represents the total time each process spent on I/O operations. Message latency is the time each message takes to reach its destination from the source, whether it's a communication or I/O message. Link traffic denotes the amount of data in bytes passing through each router. Link saturation time is defined as the total duration during which a link has exhausted all its buffers. We classify the links into local links and global links.

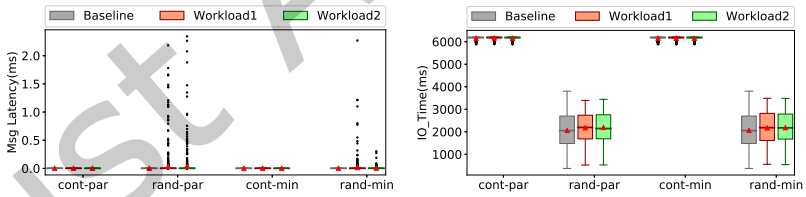
7.5 Experimental Setup

For each application, we first gather its *baseline* performance by simulating each application independently, without any other jobs sharing the network. Subsequently, we simulate the two *hybrid workloads*, collect performance metrics for each application, and compare them with the baseline cases. Each of these simulations is conducted with four different combinations of job placement policies and routing mechanisms. Each application rank is assigned to one compute node, as our focus is on network interference analysis.

The experiments are conducted on the Bebop machine at Argonne National Laboratory [9]. Bebop is equipped with 1,024 nodes, including 664 Intel Broadwell nodes and 352 Knights Landing nodes. Each Broadwell node contains a 36-core processor with 128 GB of DDR4 RAM. All of our experiments use the optimistic parallel model in CODES/ROSS and are executed on 1 Broadwell node with 36 MPI threads. The average simulation runtime is approximately 4 hours for a single run.

8 Experimental Results

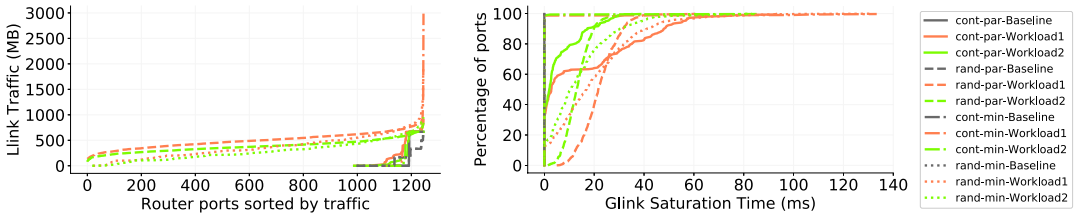
In this section, we showcase the experimental results and illustrate the co-analysis of MPI and I/O traffic interference. To evaluate the effect of network interference, we conducted a comparative analysis of performance metrics for each application across hybrid workload scenarios and baseline cases.



(a) Checkpoint I/O Msg Latency (b) Checkpoint I/O Time

Fig. 10. Message latency (left) and I/O time (right) in boxplots of Checkpoint with different configurations. Each box represents the minimum, first quartile, median, third quartile, and maximum, from bottom to top. Red triangles indicate the means.

The Checkpoint application involves negligible communication operations; thus, the message latency, as shown in Figure 10(a), can be treated as I/O message latency. Across all setups, the majority of I/O messages achieve latency of less than 2 microseconds. However, compared with baseline cases, the latency of some I/O messages is exponentially delayed due to network interference under random placement cases. Nevertheless, the total I/O time remains at the same level, as depicted in Figure 10(b). Notably, network interference does not significantly increase the total I/O time of Checkpoint.

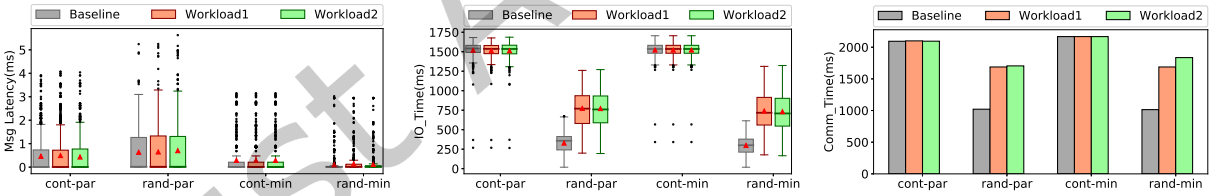


(a) Checkpoint Local Link Traffic (b) Checkpoint Global Link Saturation

Fig. 11. Local link traffic and global link saturation time of the Checkpoint application. Different curves represent various placement and routing configurations.

Figure 11 illustrates the local link traffic and global link saturation status of Checkpoint. As shown in Figure 11(a), random placement aids in distributing the substantial I/O load across a larger number of routers, whereas contiguous placement confines I/O traffic within a smaller range of routers. A significant portion of the I/O traffic is confined within a few router ports in contiguous placement, resulting in degraded I/O time, as observed in Figure 10(b). Minimal routing exacerbates this situation as it is unable to avoid hot links by routing to non-minimal paths.

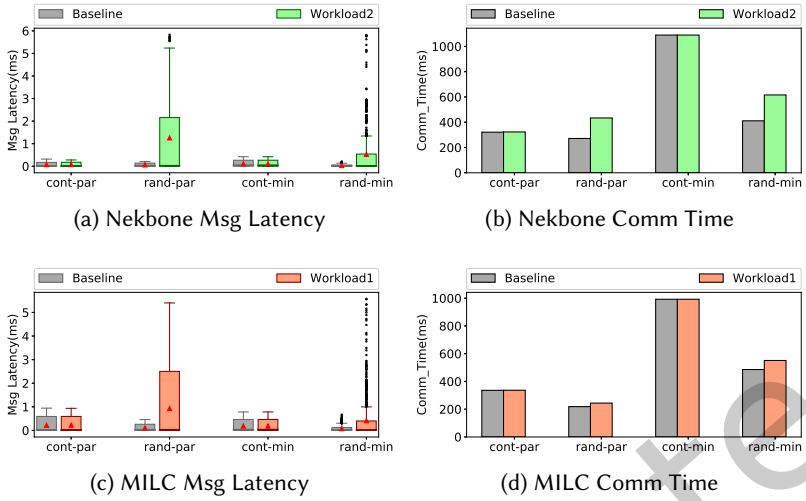
Comparing hybrid workload cases with baselines, router traffic increases slightly due to network interference. In Figure 11(b), global link saturation for baseline cases is not observed since I/O nodes tend to schedule I/O traffic to local burst buffer nodes. In cont-min cases, fewer than 2 global links experience increased saturation time of up to 106 ms due to network interference. In cont-par cases, 60% of global links experience increased saturation time due to network interference. In random placement cases, most global links experience significant increments in saturation time, indicating that random placement is sensitive to network interference.



(a) Msg Latency (b) I/O Time (c) Comm Time

Fig. 12. Message latency, I/O time, and communication time of Cosmoflow with different configurations on dragonfly system. Different colors represent different workloads including baseline.

The Cosmoflow application entails both I/O and communication loads. Each process reads a batch of samples and communicates gradients to all others every epoch. The message latency, as shown in Figure 12(a), does not distinguish between I/O traffic and communication messages. There is a negligible difference between latency distributions for baseline cases and hybrid workload cases. However, the total I/O time and communication time are significantly prolonged under random placement due to network interference. The average I/O times increase to around 750 ms for hybrid workload cases, while they remain around 300 ms for baseline cases. Similarly, the communication times for Cosmoflow experience around a 1.6x to 1.8x slowdown due to network interference under random placement.



(a) Nekbone Msg Latency (b) Nekbone Comm Time

(c) MILC Msg Latency (d) MILC Comm Time

Fig. 13. Message latency and communication time of Nekbone and MILC under different configurations.

On the other hand, comparing different placement policies, random placement results in lower absolute values for baseline cases but suffers more variability from network traffic. Conversely, contiguous placement has a higher total time but is almost immune to network interference.

Nekbone and MILC are applications with negligible I/O operations. Figure 13 displays their message latency distribution and total communication time. With random placement, both Nekbone and MILC experience degraded message latency compared to baseline cases, as shown in Figure 13(a). The average message latency increases up to 103x and 16x for Nekbone and MILC, respectively. With contiguous placement, the message latency distribution remains unaffected for both applications. In terms of total communication time, random placement suffers slowdown due to network interference, unlike contiguous placement. Nekbone shows more communication time variability than MILC since Nekbone is less communication-intensive.

Both I/O traffic and communication traffic are affected by network sharing. Applications with relatively heavy I/O loads exhibit less sensitivity to network interference in terms of I/O time. For example, Checkpoint experiences less slowdown in I/O time compared to Cosmoflow in this study. Similarly, communication time performance demonstrates a similar “bully". effect. Therefore, characterizing application communication and I/O patterns is crucial for making intelligent decisions regarding the scheduling of compute and network resources in multi-tenant systems.

The job placement method significantly impacts both I/O and communication performance. Random placement achieves better I/O and communication time but is susceptible to considerable degradation from network interference. Contiguous placement, on the other hand, results in delayed total I/O and communication time but is stable against network interference. Therefore, smart and flexible placement methodologies are needed to balance the trade-offs between random placement and contiguous placement.

In summary, network sharing and job placement significantly affect both I/O and communication performance. While adaptive routing and random placement increase the risk of network interference, contiguous placement helps mitigate it but may introduce delays. Applications with heavy I/O loads exhibit lower sensitivity to interference, though both I/O and communication traffic display “bully” effects under congestion. Random

placement can offer performance advantages but is more prone to interference, whereas contiguous placement reduces interference but may limit performance gains.

9 Conclusion

With the increasing adoption of heterogeneous architectures in HPC systems and the diverse nature of HPC workloads, understanding network interference among hybrid workloads becomes crucial. The proposed MFNetSim, a multi-fidelity modeling framework, facilitates the co-analysis of MPI and I/O traffic on modern interconnect networks by combining different levels of abstraction. This approach efficiently models communication and I/O traffic on HPC systems equipped with flash-based storage, striking a balance between detailed, high-fidelity network models and more efficient, low-fidelity storage models.

Our multi-fidelity design addresses critical trade-offs between accuracy and scalability. A fully high-fidelity framework would be computationally prohibitive for large systems. Conversely, a fully low-fidelity framework would sacrifice network accuracy, rendering interference analysis unreliable. MFNetSim's hybrid design ensures scalable yet precise modeling of network contention, enabling practical studies of multi-tenant systems.

Through large-scale simulation studies of hybrid workloads comprising traditional HPC applications and emerging ML applications, we found that both I/O and communication traffic are susceptible to network interference. Different job placement methods offer trade-offs between stable performance and fast performance. Our future work will involve expanding the range of applications, aiming to provide a more comprehensive understanding of network interference exhibited by various types of applications.

While this work models communication and I/O traffic, it is primarily tailored to HPC systems with flash-based storage, limiting its generalizability to other storage architectures. Additionally, the applications examined may not comprehensively represent the diverse workloads on Dragonfly systems. Future research should include a broader range of applications to better understand network interference in different workload scenarios.

MFNetSim has been released as open-source software on GitHub, offering a reproducible tool for the parallel discrete-event simulation and systems research communities to study workload interference in large-scale systems [20].

Acknowledgments

This work is supported in part by the U.S. Department of Energy through Contract DE-SC0024271 under the Tachyon project, and by the U.S. National Science Foundation under grants OAC-2402901 and CCF-2413597.

References

[1] ALCF. 2024. *Aurora.* http://aurora.alcf.anl.gov
[2] ALCF. 2024. *Theta.* https://www.alcf.anl.gov/theta
[3] DAOS. 2024. *DAOS Online Document.* https://daos.io/daos-overview
[4] Nan Jiang, Daniel U Becker, George Michelogiannakis, James Balfour, Brian Towles, David E Shaw, John Kim, and William J Dally. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 86–96.
[5] Yao Kang, Xin Wang, and Zhiling Lan. 2021. Q-adaptive: A multi-agent reinforcement learning based routing on dragonfly network. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. 189–200.
[6] Yao Kang, Xin Wang, Neil McGlohon, Misbah Mubarak, Sudheer Chunduri, and Zhiling Lan. 2019. Modeling and analysis of application interference on dragonfly+. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 161–172.
[7] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.
[8] Julian M Kunkel. 2013. Simulating parallel programs on application and system level. *Computer Science-Research and Development* 28 (2013), 167–174.
[9] LCRC. 2024. *Bebop.* http://www.lcrc.anl.gov/systems/resources/bebop

[10] Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Kärnä, Diana Moise, Simon J Pennycook, et al. 2018. CosmoFlow: using deep learning to learn the universe at scale. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 819–829.

[11] Misbah Mubarak, Philip Carns, Jonathan Jenkins, Jianping Kelvin Li, Nikhil Jain, Shane Snyder, Robert Ross, Christopher D Carothers, Abhinav Bhatele, and Kwan-Liu Ma. 2017. Quantifying i/o and communication traffic interference on dragonfly networks equipped with burst buffers. In *IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 204–215.

[12] Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Philip Carns. 2017. Enabling parallel simulation of large-scale HPC network systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2017), 87–100.

[13] Misbah Mubarak, Neil McGlohon, Malek Musleh, Eric Borch, Robert B Ross, Ram Huggahalli, Sudheer Chunduri, Scott Parker, Christopher D Carothers, and Kalyan Kumaran. 2019. Evaluating Quality of Service Traffic Classes on the Megafly Network. In *International Conference on High Performance Computing*. Springer, 3–20.

[14] Misbah Mubarak and Robert B. Ross. 2017. *Validation Study of CODES Dragonfly Network Model with Theta Cray XC System.* Technical Report. Argonne National Laboratory. http://www.mcs.anl.gov/publication/validation-study-codes-dragonfly-network-model-theta-cray-xc-systemser

[15] S. Pakin. 2004. coNCePTuaL: a network correctness and performance testing language. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* 79–. doi:10.1109/IPDPS.2004.1303014

[16] S. Pakin. 2007. The Design and Implementation of a Domain-Specific Language for Network Performance Testing. *IEEE Transactions on Parallel and Distributed Systems* 18, 10 (Oct 2007), 1436–1449. doi:10.1109/TPDS.2007.1065

[17] Benjamin S Parsons and Vijay S Pai. 2013. A mathematical hard disk timing model for full system simulation. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 143–153.

[18] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. 2011. The structural simulation toolkit. *SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[19] Chris Ruemmler and John Wilkes. 1994. An introduction to disk drive modeling. *Computer* 27, 3 (1994), 17–28.

[20] Xin Wang, Kevin A. Brown, Robert B. Ross, Christopher D. Carothers, and Zhiling Lan. 2025. *MFNetSim: A Multi-Fidelity Network Simulation Framework for Multi-Traffic Modeling of Dragonfly Systems.* https://github.com/SPEAR-UIC/MFNetSim.git

[21] Xin Wang, Misbah Mubarak, Yao Kang, Robert B Ross, and Zhiling Lan. 2020. Union: An automatic workload manager for accelerating network simulation. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 821–830.

[22] X. Wang, M. Mubarak, X. Yang, R. B. Ross, and Z. Lan. 2018. Trade-Off Study of Localizing Communication and Balancing Network Traffic on a Dragonfly System. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1113–1122. doi:10.1109/IPDPS.2018.00120

[23] Xu Yang, John Jenkins, Misbah Mubarak, Robert B. Ross, and Zhiling Lan. 2016. Watch out for the Bully!: Job Interference Study on Dragonfly Network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 64, 11 pages. http://dl.acm.org/citation.cfm?id=3014904.3014990

[24] X. Yang, J. Jenkins, M. Mubarak, X. Wang, R. B. Ross, and Z. Lan. 2016. Study of Intra- and Interjob Interference on Torus Networks. In *IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 239–246. doi:10.1109/ICPADS.2016.0040

[25] Jinsoo Yoo, Youjip Won, Sooyong Kang, Jongmoo Choi, Sungroh Yoon, and Jaehyuk Cha. 2014. Analytical model of SSD parallelism. In *2014 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*. IEEE, 551–559.

Received 18 February 2025; revised 18 February 2025; accepted 8 April 2025