Evaluating the Network Effects of Orchestration Strategies for AI Workloads in Modern Data Centers

José Santos*, Pavlos Maniotis†, Chen Wang†, Asser Tantawi†, Olivier Tardieu†, Tim Wauters*, Filip De Turck*
* Ghent University - imec, IDLab, Department of Information Technology, Gent, Belgium
Email: {josepedro.pereiradossantos, tim.wauters, filip.deturck}@UGent.be
† IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA

***Abstract*—The exponential growth in Artificial Intelligence (AI) adoption presents unique challenges and opportunities for deploying AI workloads in modern Data Center (DC) networks, particularly in terms of performance, scalability, and reliability. AI workloads, such as inference and distributed training, impose different network demands: inference is primarily compute-bound and typically requires low network latency, while distributed training is network-bound and requires high bandwidth, placing significant strain on the network. This paper focuses on the network requirements of widely known AI communication patterns, and studies their impact on modern DC architectures by analyzing the effects of different orchestration strategies—specifically *packing* and *spreading*—on throughput, response time, and network congestion. The results show that *packing* strategies generally deliver higher performance for most covered AI collectives. However, *spreading* strategies can be beneficial in certain scenarios, such as when larger workloads span across higher number of racks, as they can help mitigate network congestion between the switches of leaf-spine network configurations. This paper offers valuable insights into optimizing the orchestration of popular AI collectives in data center networks, presenting informed strategies to improve performance in response to growing AI demands, with findings demonstrating completion time reductions of up to 30%.**

***Index Terms*—Artificial Intelligence, Data Center, Orchestration, Performance, Simulation**

I. Introduction

The rapid growth of Artificial Intelligence (AI) applications has fundamentally transformed the demands on modern Data Center (DC) networks [1]. From autonomous systems to large-scale natural language processing models, AI workloads are becoming increasingly diverse and complex, presenting unique challenges and opportunities for modern DCs [2]. These workloads impose strict requirements for performance, scalability, and reliability, necessitating innovative strategies to optimize their orchestration within DC environments [3].

Recent advances in AI have led to an increased interest in Generative Artificial Intelligence (GenAI) systems, such as Foundation Models (FMs) [4] and Large Language Models (LLMs) [5]. This has driven exponential growth in both the size of training datasets and the number of model parameters, as illustrated in Fig. 1. Industry best practices now commonly involve models with millions to billions of parameters [6], [7], with recent breakthroughs extending into the trillion-parameter range [8]. As a result, highly parallelized AI training has become standard practice, involving a plethora of bandwidth-hungry tasks operating in synchrony, which demand high-speed networks to efficiently interconnect the constantly increasing number of computing chips [9].

AI workloads can be broadly classified into inference and training tasks [10]. Inference tasks are typically compute-bound, requiring efficient processing capabilities and low latency networks to deliver real-time predictions. In contrast, training tasks are mostly network-bound, depending on high bandwidth links and switches to facilitate the exchange of model parameters across multiple servers. These requirements present significant challenges in efficient orchestration in modern DC environments, as a mix of both AI and non-AI applications is run by multiple users. On one hand, network-bound workloads, such as distributed training for LLMs [11], require Remote Direct Memory Access (RDMA) networks and network locality to meet their requirements, while other reliability-focused workloads, such as storage, can function more effectively over Transmission Control Protocol (TCP) and more diverse orchestration strategies [12]. This trade-off complicates orchestration decisions, making it challenging to identify the most appropriate strategy each time.

This paper studies the impact of widely used AI communication patterns, such as those from *all-to-all* and *all-gather* operations, on modern DC architectures [13], providing a comprehensive analysis of their performance characteristics. We explore the trade-offs between *packing* and *spreading* orchestration strategies, highlighting scenarios where each approach proves advantageous. This work builds on our previous work [14], where we introduced *Chic-sched*, a placement-group scheduler designed for distributed workloads on hierarchical topologies. *Chic-sched* operates with loosely defined constraints, such as *packing* and *spreading*, and avoids retries by offering suboptimal orchestration strategies that reduce placement failures. In this work, we evaluate the impact of *Chic-sched* strategies on the communication overhead of common collective operations used in modern AI workloads.

The main contributions of this paper are twofold:

- **Performance Evaluation with Popular AI Communication Patterns**: We study the performance of various orchestration strategies based on *packing* and *spreading*. The study considers a model of a modern DC infrastructure inspired by IBM Vela [13], a supercomputing platform integrated into the IBM Cloud and optimized for AI workloads. Vela provides scalable, dynamic, multi-

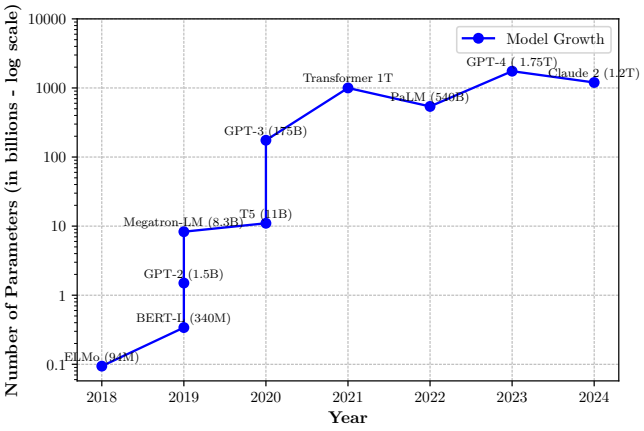


Fig. 1: The evolution of GenAI workloads over recent years, showing the increase in model sizes measured by the number of parameters (in billions) [15].

tenant, and geographically distributed infrastructure for large-scale model training and other AI workflow processes. The results indicate that *packing* workloads within a single rack generally leads to higher performance, especially in terms of completion/execution time. However, *spreading* workloads across multiple racks can be advantageous in certain scenarios as it can help mitigate network congestion between switches of leaf-spine network configurations. In many cases, the proper orchestration strategy can reduce the execution time of AI collectives by up to 30%.

- **Valuable Insights for Developers and DC Architects:** By exploring diverse orchestration strategies, this paper highlights key factors that influence the performance and efficiency of DC networks in supporting demanding AI workloads. These insights are valuable in understanding how DCs can be adapted to handle the growing demands of future workloads, ensuring that the infrastructure is scalable, efficient, and robust for a wide variety of deployment environments.

The remainder of this paper is organized as follows: Section II reviews prior studies on AI and DC design, focusing on the performance impact of these highly demanding workloads. Section III details the methodology used, including the simulation framework and the associated parameters. Section IV describes the experimental setup, while Section V presents the results, discusses the main findings, and highlights the trade-offs between various placement strategies. Finally, Section VI concludes the paper and suggests directions for future research.

II. Related Work

Data Center (DC) Networking has evolved to meet the growing demands of AI workloads, particularly those requiring high bandwidth and low latency. Previous studies [16], [17], [18] have highlighted the role of advanced interconnect technologies, such as RDMA and InfiniBand, in facilitating efficient communication between servers. These technologies are crucial for network-bound workloads, such as distributed AI training, where timely exchange of model parameters is essential. Efforts to optimize network architectures have focused on mitigating congestion and ensuring predictable latency [19], [20], [21]. In addition, other studies have shown that topology-aware routing and load balancing mechanisms can significantly improve network performance [22], [23], [24]. More recently, disaggregated DCs, combined with optical connections, have been proposed to improve scalability and reduce power consumption [25], [26].

AI Application Modeling is essential for understanding the unique demands that AI applications place on DC networks. Recent studies [27], [28], [29], [30] have investigated the impact of AI communication patterns on network performance. However, such studies often rely on simplified formulations for the networking component and offer limited support for integrating more advanced network simulators, such as ns-3 [31] and OMNeT++ [32]. To analyze the orchestration of AI applications efficiently, both their communication patterns and the DC topology must be considered, as understanding them is key to reducing network overhead and minimizing workload execution time.

This paper builds upon prior studies by analyzing the interplay between common AI communication patterns and widely used orchestration strategies in modern DC infrastructures, evaluating their impact on application performance. These insights can help optimize AI workload orchestration in current and next-generation cloud environments.

III. Methodology

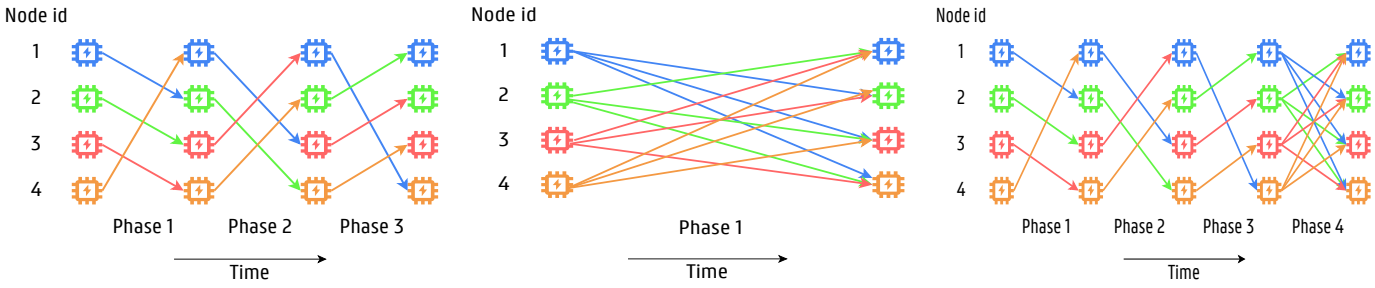
This section presents the methodology used to assess the performance of AI workloads using a discrete-event network simulator. The simulation parameters employed in the experiments are detailed in Section IV.

A. Discrete-Event Simulator Overview

In our experiments, we use the *Venus* discrete-event network simulator [33], which is developed on top of the queue-based OMNEST simulation framework [34]. *Venus*, developed within IBM Research, has been widely utilized in various projects over the years [35], [36]. This study leverages *Venus* to evaluate the performance of various collective operations under different orchestration strategies, focusing on *packing* and *spreading* approaches. The considered collectives exhibit communication patterns commonly encountered in both AI workloads and other scientific parallel applications, and they are described in the next section in detail. The analysis covers both fixed- and varying-sized collectives, executed across different number of racks and under different orchestration schemes. This includes scenarios in which combinations of two AI collective operations are executed simultaneously. The goal is to identify optimal orchestration strategies that maximize throughput and minimize completion time.

B. AI Communication Patterns

The communication patterns of collective operations play a significant role in the performance of AI and scientific work-



(a) *All-to-All* (b) *All-Gather* (c) *All-Reduce*

Fig. 2: A schematic overview of the evaluated AI communication patterns for 4 communicating nodes.

loads, especially in large-scale data exchange and synchronization scenarios. This study considers three popular collective operations, as illustrated in Fig. 2. These patterns represent both single- and multi-phased operations and include *All-to-All* (Fig. 2a), *All-Gather* (Fig. 2b), and *All-Reduce* (Fig. 2c). It is important to note that the exact network footprint of each operation is highly dependent on the specific implementation of the collective communication library. The examples shown in Fig. 2 are used solely for the purposes of this analysis and are not intended to represent a one-to-one correspondence with any specific library implementation.

All-to-All involves every node in the application exchanging data with all other nodes. This pattern is commonly used in distributed training and data-parallel computations where intermediate results need to be shared among all participants. The communication overhead grows with the number of participating nodes, making it particularly sensitive to network topology and orchestration strategies. From a single-node perspective, *All-to-All* follows a deterministic sequence of $N-1$ phases, where N represents the total number of participating nodes. Each successive phase involves communication between each node and a different destination node, resulting in a total of $N \times (N-1)$ messages exchanged.

All-Gather involves each node collecting data from all other nodes, so that after the operation, each node has the full dataset. This pattern is frequently found in AI workloads, such as parameter synchronization in distributed deep learning after gradient updates, and is used in popular Deep Learning (DL) frameworks like TensorFlow and PyTorch [37]. The performance of this operation is highly dependent on the efficiency of multiple parallel data transfers between nodes.

All-Reduce consists of two distinct phases: *reduction* and *broadcast*. In the reduction phase, each node contributes its data for a specified reduction operation (e.g., sum, min, max), after which the result is broadcast to all nodes. The reduction phase typically occurs at a central point, but it may also be performed in a distributed manner across all nodes. In this study, from a communication patterns perspective, we consider the combination of *All-to-All* followed by *All-Gather* as a method for approximating the data flow of *All-Reduce*. By first simulating the *All-to-All* pattern, we distribute the data across the system, simulating a distributed reduction. Then, by simulating *All-Gather*, we effectively collect the distributed

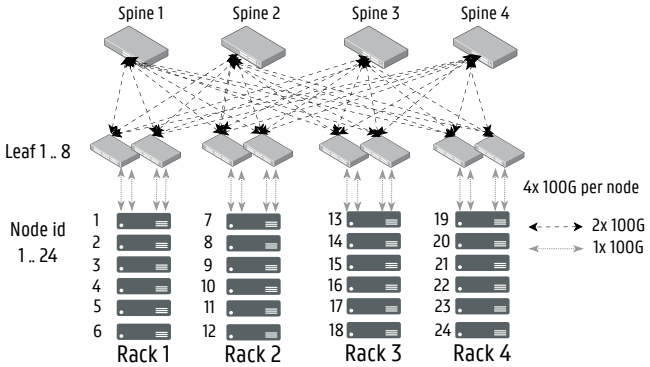


Fig. 3: An overview of the DC architecture modeled in the *Venus* simulator [13].

data at each node, ultimately providing the final aggregated result [38]. This communication pattern is critical for tasks such as gradient synchronization in distributed model training, data aggregation in distributed databases, and the efficient combination and analysis of data in big data analytics.

These communication patterns are essential for understanding the impact of different orchestration strategies on workload performance. By analyzing them in conjunction with a detailed DC infractucture model, we aim to identify efficient orchestration methods that minimize communication overhead and optimize the performance of distributed AI and scientific workloads.

IV. Evaluation Setup

This section provides an overview of the modeled DC infrastructure, along with the evaluated scenarios and orchestration strategies.

A. Data Center (DC) Infrastructure Model

Fig. 3 provides an overview of the DC infrastructure modeled in the *Venus* simulator, inspired by *Vela*, a horizontally scalable DC supercomputer built by IBM that incorporates a two-layer leaf-spine network. In 2023, IBM released architectural details and design principles behind *Vela*, which is the first cloud-native AI supercomputer seamlessly integrated into IBM Cloud [39]. *Vela* is designed with flexibility and scalability at its core and is capable of training today's large-scale GenAI models while remaining adaptable to future

demands [13]. As shown in Fig. 3, each compute node in our model is connected to two leaf switches via four 100G network interfaces. The network comprises eight leaf switches and four spine switches, arranged in a two-level Clos topology with Equal-Cost Multi-Path routing (ECMP) routing. Each leaf switch has a 2:3 oversubscription ratio and is connected to each spine via two 100G links. This results in a total rack bandwidth of 1.6 Tbps per direction as there are two leaf switches per rack. This configuration ensures uninterrupted operation in the event of Network Interface Card (NIC), leaf, or spine switch failures. The model consists of a total of 24 servers, arranged in four racks of six servers in each. It should be noted that while our model is inspired by Vela, it is not intended to be a one-to-one representation, as Vela is a larger system, and not all of its components are modeled in *Venus*, such as the Graphics Processing Units (GPUs) and the full Software-Defined Networking (SDN) stack.

B. AI Collective Scenarios

TABLE I: Evaluated AI collectives based on their size.

Name	Message Size (MB)	AI pattern	Description
4n	[1.0, 10.0, 100.0]	[*All-to-All*, *All-Gather*, *All-Reduce*]	The operation runs on 4 nodes.
6n			The operation runs on 6 nodes.
8n			The operation runs on 8 nodes.
12n			The operation runs on 12 nodes.
4n_6n			Two operations simultaneously: 1^{st} on 4 nodes and 2^{nd} on 6.
6n_8n			Two operations simultaneously: 1^{st} on 6 nodes and 2^{nd} on 8.

TABLE II: Evaluated orchestration strategies for the AI collectives presented in Table I.

Orch. Strategy	Node Identifiers (Ids)
4n_1r	[1, 2, 3, 4]
4n_2r	[1, 2, 7, 8]
4n_3r	[1, 2, 7, 8]
4n_4r	[1, 7, 13, 19]
6n_1r	[1, 2, 3, 4, 5, 6]
6n_2r	[1, 2, 3, 7, 8, 9]
6n_3r	[1, 2, 7, 8, 13, 14]
6n_4r	[1, 2, 7, 8, 13, 19]
8n_2r	[1, 2, 3, 4, 7, 8, 9, 10]
8n_3r	[1, 2, 3, 7, 8, 9, 13, 14]
8n_4r	[1, 2, 7, 8, 13, 14, 19, 20]
12n_2r	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
12n_3r	[1, 2, 3, 4, 7, 8, 9, 10, 13, 14, 15, 16]
12n_4r	[1, 2, 3, 7, 8, 9, 13, 14, 15, 19, 20, 21]
4n_1r_6n_1r	[1, 2, 3, 4] & [7, 8, 9, 10, 11, 12]
4n_2r_6n_2r	[1, 2, 7, 8] & [13, 14, 15, 19, 20, 21]
4n_3r_6n_3r	[1, 2, 7, 13] & [8, 9, 14, 15, 19, 20]
4n_4r_6n_4r	[1, 7, 13, 19] & [2, 3, 8, 9, 14, 20]
6n_1r_8n_2r	[1, 2, 3, 4, 5, 6] & [7, 8, 9, 10, 13, 14, 15, 16]
6n_2r_8n_2r	[1, 2, 3, 7, 8, 9] & [13, 14, 15, 16, 19, 20, 21, 22]
6n_3r_8n_3r	[1, 2, 7, 8, 13, 14] & [9, 10, 11, 15, 16, 17, 19, 20]
6n_4r_8n_4r	[1, 2, 7, 8, 13, 19] & [3, 4, 9, 10, 14, 15, 20, 21]

Table I summarizes the scenarios evaluated in this work. Each scenario is characterized by the number of nodes involved in the operation, the message size exchanged, and the communication pattern. These collective operations are designed to represent a broad spectrum of scenarios commonly encountered in distributed AI workloads. Each pattern is simulated using message sizes of 1 MB, 10 MB, and 100 MB to represent varying data transfer requirements. In all scenarios, the packet size is fixed at 1 KiB, meaning that larger messages are segmented into multiple contiguous packets. These scenarios provide a comprehensive analysis of the interaction between different communication patterns, message sizes, and node counts, offering valuable insights into the performance of typical AI collectives under various orchestration strategies.

C. Orchestration Strategies

Table II summarizes the orchestration strategies used in our evaluation. These strategies vary based on how the workloads are distributed across racks, simulating a wide range of deployment scenarios in modern DC environments. The placement strategies are categorized based on the number of nodes involved in the operations (i.e., 4, 6, 8, or 12 nodes) and their distribution across one or more racks. For example, the *4n_1r* strategy uses four nodes co-located within a single rack, minimizing communication overhead and simulating a compact deployment. In contrast, the *6n_3r* strategy distributes six nodes across three distinct racks, increasing inter-rack communication and mimicking a more dispersed workload. The last group of strategies represents scenarios in which two collectives run concurrently. For instance, *6n_1r_8n_2r* represents one collective executing on six nodes within a single rack, while the second collective operates across eight nodes distributed over two racks. It should be noted that the term *node* does not refer to a server, but rather to a virtual entity with a single or multiple network interfaces, such as a container or Virtual Machine (VM), executing the workload on a physical server. Thus, each server may use one or more network interfaces during the simulation of an operation, depending on the type or number of nodes placed on it, a detail that is described further on a per-scenario basis. Each strategy is designed to assess the communication overhead, which directly affects overall workload performance:

- **Single-Rack Placement**: All nodes are placed in a single rack, ensuring that only intra-rack communication takes place. For example, the *4n_1r* strategy places all four nodes on separate servers within the same rack.
- **Multi-Rack Placement**: Nodes are distributed across two, three, or four racks, introducing varying levels of inter-rack communication overhead. For example, in the *4n_4r* strategy, the four nodes are placed on separate servers in distinct racks, maximizing inter-rack communication.
- **Concurrent Collectives**: Some strategies involve running two collectives simultaneously, each with a different node count and placement configuration. For example, the *4n_1r_6n_1r* strategy runs a 4-node collective on four servers in one rack and a 6-node collective on six servers in another rack.

Each strategy is simulated 30 times to ensure statistical significance, with the results presented with a confidence

interval of 95%. The entire simulation process, which includes all strategies and collectives, was fully automated using custom shell scripts.

D. Performance Metrics

In modern DC networks, the performance of collective operations is strongly influenced by the underlying communication patterns and orchestration strategies. In this study, we evaluate performance using the following key metrics:

- **Throughput (in Gbit/s)**: Measures the rate at which packets are transmitted across the network, expressed in gigabits per second. The throughput per interface (i.e., determined by the number of network endpoints involved in the collective operation) is also provided.
- **Completion Time (in ms)**: Measures the time required for the collective operation to complete, expressed in milliseconds. This includes all communication delays, such as queuing, transmission, and propagation delays.
- **Number of Hops**: Used to evaluate the distribution of packets based on the number of hops they traverse across the network. A hop refers to the transmission of a packet through a network switch. The analysis differentiates between packets that travel:
  - **1 hop** : Communication within the same rack, i.e., intra-rack communication
  - **3 hops** : Communication between different racks, i.e., inter-rack communication.

These metrics provide valuable insights into the network's ability to handle the demands of the simulated operations, helping to identify potential bottlenecks or inefficiencies caused by the chosen orchestration strategy.

V. Results

A. All-to-All

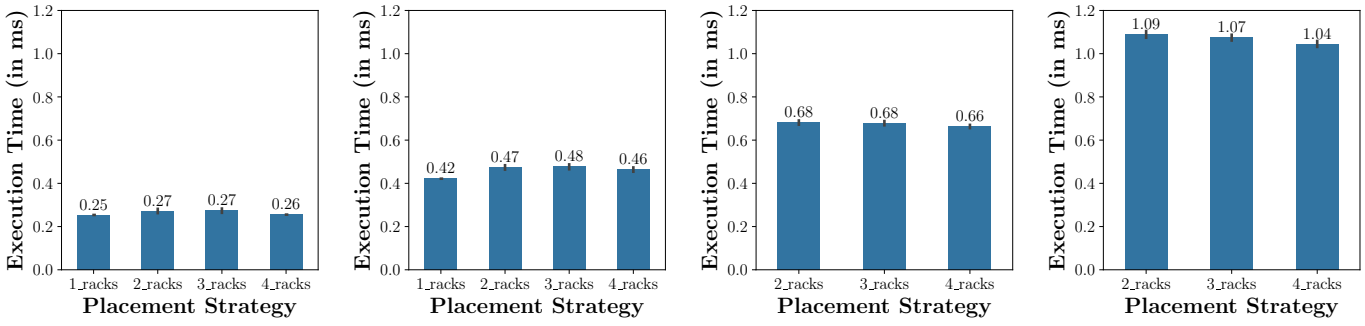
Fig. 4 shows the completion time, i.e., execution time, of the *All-to-All* operation and demonstrates its sensitivity to message size, orchestration strategy, and number of nodes, with each participating node using a single server network interface. For a message size of 1 MB, the collective's execution time remains relatively bounded, even as the number of nodes and racks increases. However, configurations with more racks consistently exhibit higher execution times due to network contention for the shared link resources required for inter-rack communication, highlighting the importance of an optimized placement strategy. In contrast, for a message size of 10 MB, the execution times increase significantly, particularly in configurations with more nodes and racks. This increase is driven by the larger data volumes that are exchanged between the nodes, which contribute to greater network congestion and communication overhead. For larger collectives that do not fit within a single rack (assuming one node per server), such as those with eight or twelve nodes, distributing them across multiple racks appears advantageous. Considering the oversubscription ratio at the leaf layer, distributing the nodes across multiple racks provides higher inter-node bandwidth overall, which helps better balance the traffic toward the spines and reduces congestion and delays compared to scenarios with fewer racks. Fig. 5 shows the measured throughput during the execution of the operation for a message size of 10 MB, revealing a trend similar to that of the execution time. When the job does not fit within a single rack, strategies that effectively distribute the workload across multiple racks perform slightly better.

When the nodes are located within a single rack, execution times are significantly lower and more consistent, even for larger message sizes. This is shown in the results of Fig. 6, which considers the additional scenarios in Table III. In contrast to the scenarios of Fig. 4, these scenarios utilize multiple interfaces per server, as multiple single-interface nodes are co-located on the same server. For example, in the *4n_3_1* strategy, there are three network interfaces used on server 1, and one network interface used on server 2. For four-node scenarios without spine-crossing, execution times remain uniform across runs, as traffic avoids the variability introduced by network contention for shared resources. In other words, ECMP collisions are eliminated since no traffic crosses the spines. In cases where spine crossing occurs—i.e., when traffic moves between switches within the same rack—execution time increases slightly and varies across runs, as network contention cannot be completely eliminated. Increasing to six nodes within the same rack results in further increase in execution time, mainly due to the higher volume of data that needs to be exchanged. Compared to multi-rack strategies, performance remains constantly better.

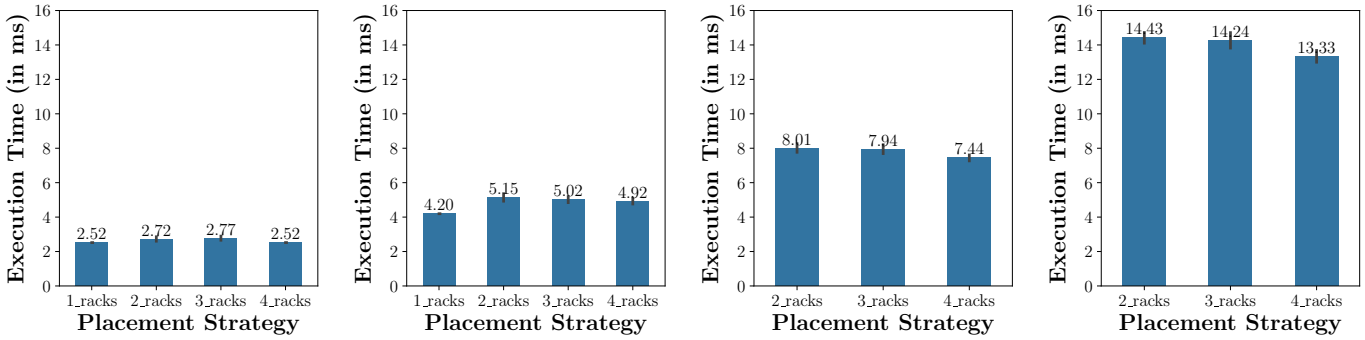
When multiple collectives run concurrently, *packing* each collective into a dedicated rack proves beneficial, as illustrated in Fig. 7. This strategy reduces interference between AI collectives, allowing each to take advantage of localized communication. These findings suggest that a balanced placement strategy—whether *spreading* or *packing*—tailored to job size and the characteristics of concurrent collectives is essential for optimizing performance.

B. All-Gather

The *All-Gather* communication pattern has been evaluated for two cases: one where each participating node uses a single server network interface (Figures 8a - 8d), and another where each node uses all four server network interfaces (Figures 8e - 8h). These scenarios are run for a message size of 100 MB (injected per interface). In the single-interface case, the results show minimal variation across different placement strategies and collective sizes. This suggests that, in this configuration, placement strategies have a minimal impact on the operation's performance. Whether the nodes are packed within a single rack or spread across multiple racks, execution time remains relatively consistent. In contrast, when all four interfaces are used by the nodes, significant differences are observed, mirroring the trends seen in the *All-to-All* pattern. For smaller collective operations (i.e., 4- and 6-node configurations), packing within a single rack minimizes the execution time, resulting in execution time reductions of 10% to 30%. However, as the size

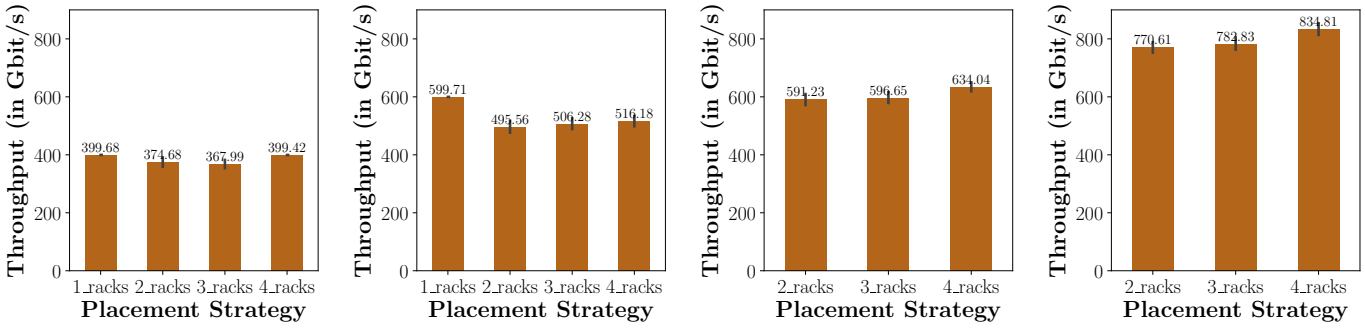


(a) 4-nodes - 1 MB. (b) 6-nodes - 1 MB. (c) 8-nodes - 1 MB. (d) 12-nodes - 1 MB.



(e) 4-nodes - 10 MB. (f) 6-nodes - 10 MB. (g) 8-nodes - 10 MB. (h) 12-nodes - 10 MB.

Fig. 4: Execution time and standard deviation (in ms) for the *All-to-All* pattern.



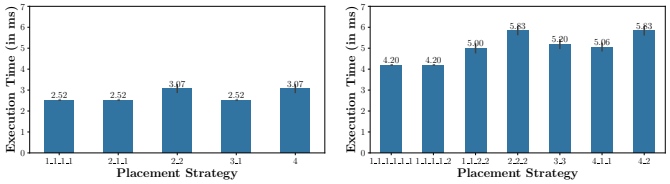
(a) 4-nodes - 10 MB. (b) 6-nodes - 10 MB. (c) 8-nodes - 10 MB. (d) 12-nodes - 10 MB.

Fig. 5: Total throughput (in Gbit/s) for the *All-to-All* pattern for a message size of 10 MB.

TABLE III: Additional strategies for workload placement within a single rack.

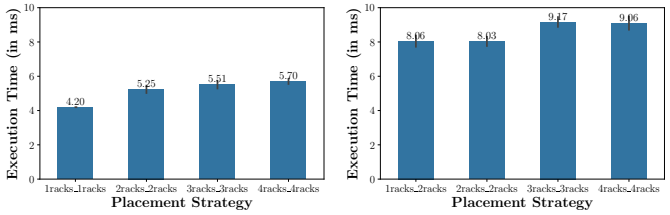
Placement Strategy	Node Identifiers (Ids)	Description
4n_1_1_1_1	[1, 2, 3, 4]	Four nodes spread on separate servers.
4n_2_1_1	[1, 1, 2, 3]	Two nodes placed on server 1 and two nodes spread on separate servers.
4n_2_2	[1, 1, 2, 2]	Two nodes places on server 1 and two nodes placed on server 2.
4n_3_1	[1, 1, 1, 2]	Three nodes placed on server 1 and one node plaed on server 2.
4n_4	[1, 1, 1, 1]	Four nodes placed on server 1.
6n_1_1_1_1_1_1	[1, 2, 3, 4, 5, 6]	Six nodes spread on separate servers.
6n_1_1_1_1_2	[1, 1, 2, 3, 4, 5]	Two nodes placed on server 1 and four nodes spread on separate servers.
6n_1_1_2_2	[1, 1, 2, 2, 3, 4]	Two nodes placed on server 1, two nodes on placed on server 2, and two spread on separate servers.
6n_2_2_2	[1, 1, 2, 2, 3, 3]	Two nodes placed on server 1, two nodes placed on server 2, and two nodes placed on server 3.
6n_3_3	[1, 1, 1, 2, 2, 2]	Three nodes placed on server 1, and three nodes placed on server 2.
6n_4_1_1	[1, 1, 1, 1, 2, 3]	Four nodes placed on server 1, one node placed on server 2, and one node placed on server 3.
6n_4_2	[1, 1, 1, 1, 2, 2]	Four nodes placed on server 1, and two nodes placed on server 2.

of the operation increases (i.e., 8- and 12-node configurations), spreading across multiple racks becomes advantageous, yielding execution time reductions of 13% to 20%. Furthermore, when two collectives are executed concurrently, placing each collective in its own dedicated rack improves performance. For brevity, the graphs illustrating these results are omitted. This strategy minimizes inter-workload interference, enabling each collective to take advantage of the network isolation.



(a) 4-nodes - 10 MB. (b) 6-nodes - 10 MB.

Fig. 6: Execution time (in ms) for the *All-to-All* pattern when all nodes are placed within a single rack.



(a) 4-nodes & 6-nodes - 10 MB. (b) 6-nodes & 8-nodes - 10 MB.

Fig. 7: Execution time (in ms) for the *All-to-All* pattern when two collectives are executed simultaneously.

C. All-Reduce

The performance of the *All-Reduce* communication pattern has been evaluated for various node counts and orchestration strategies, as shown in Table IV. In this set of experiments, each participating node uses a single server network interface. The results indicate that the orchestration strategy significantly influences execution time, similar to the *All-to-All* pattern, which is simulated as a core component of the operation during the distributed reduction phase. For smaller collectives, *packing* within a single rack consistently results in the lowest execution time. For instance, the 4-node collective achieves an execution time of 5.04 ms when packed within a single rack, compared to 5.67 ms when distributed across two racks. Similarly, the 6-node configuration shows an execution time of 8.39 ms in a single rack, compared to 10.58 ms and 10.72 ms when distributed across two and three racks, respectively. This represents a reduction in execution time of 26% and 27%, respectively, when using a single-rack orchestration strategy compared to a multi-rack strategy. Packet-level insights further explain these performance differences. In single-rack configurations, packets traverse only one hop, eliminating contention from ECMP collisions. In contrast, multi-rack deployments introduce multi-hop packets, which place additional stress on the network fabric and lead to higher communication overhead. For example, in the 6-node collective, the number of 3-hop packets increases significantly with rack count—from 0.37M in the 2-rack case to 0.53M in the 4-rack configuration—highlighting the greater burden on the network fabric.

As the collective size increases, the advantages of *spreading* workloads across multiple racks become more apparent. For example, the 8-node collective experiences a time reduction of approximately 4% when deployed across four racks, compared to being distributed across two or three racks. Considering the oversubscription ratio at the leaf layer, distributing the nodes across multiple racks provides higher inter-node bandwidth overall, which helps better balance the traffic toward the spines and reduces congestion and delays compared to scenarios with fewer racks. Moreover, when running concurrent collectives, isolating each collective within a dedicated rack leads to significant performance gains, as this strategy minimizes inter-workload interference.

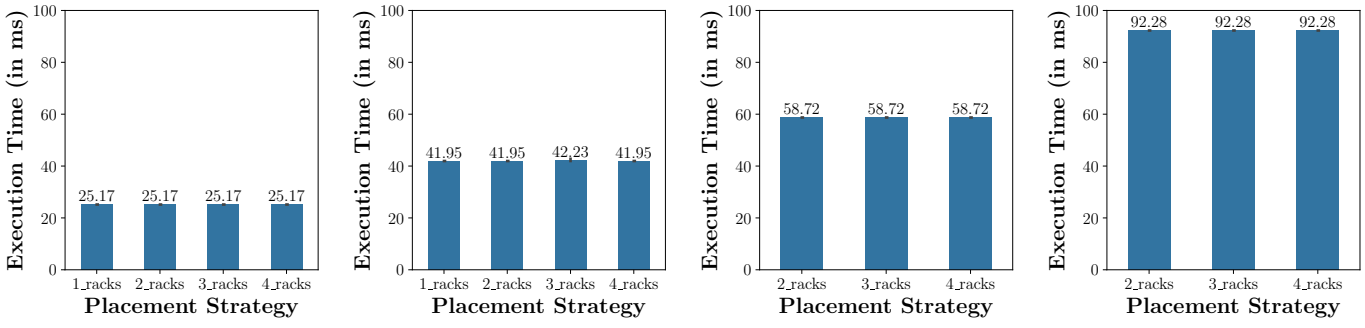
Overall, the *All-Reduce* pattern highlights the importance of efficient workload orchestration and the strategic utilization of available network resources to minimize execution times. These findings align with trends observed in the other patterns, highlighting the critical role of informed orchestration strategies in high-performance computing environments.

D. Summary

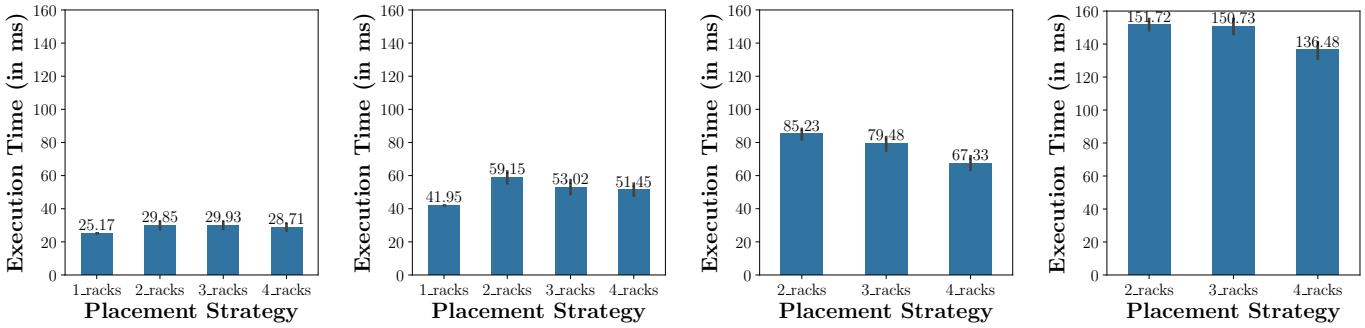
Packing collectives within a single rack typically results in higher performance, especially in reducing the collective's execution time. This is attributed to the elimination of cross-rack traffic, which benefits from reduced communication overhead and lower packet delays. In contrast, *spreading* collectives across racks introduces additional communication overhead and congestion within the fabric. However, *spreading* collectives across racks can be advantageous in specific scenarios. For instance, when a job cannot fit within a single rack (e.g., an 8- or 12-node collective, assuming one node per server), distributing nodes across multiple racks improves performance and reduces execution times. This approach helps balance traffic toward the spines more effectively, reducing congestion and delays compared to scenarios with fewer racks, particularly when the network is oversubscribed. Similarly, isolating applications in separate racks reduces network contention for shared resources, further enhancing performance. These strategies highlight the importance of tailoring placement approaches to align with workload and infrastructure characteristics for optimal results.

VI. Conclusions

This paper investigated the dynamics of deploying AI workloads in modern DC networks, focusing on the communication impact of commonly used AI collective operations and orchestration strategies. By modeling a state-of-the-art DC infrastructure inspired by IBM Vela, this study demonstrates the impact of different placement strategies on key performance metrics, including throughput, latency, and network congestion. The findings emphasize the importance of tailoring orchestration strategies to the specific characteristics of workloads. While *packing* strategies typically offer higher performance for most operations, *spreading* strategies effectively mitigate contention for larger-scale jobs that cannot fit within a single rack. These insights are valuable for developers and DC architects looking to optimize AI workload orchestration in increasingly complex cloud environments. This work contributes to the development of more efficient and robust placement strategies that can meet the evolving demands of AI applications. In multiple cases, the execution time of the collective operation can be reduced by up to 30% with the right orchestration approach. Future



(a) 4-nodes - Single Interface. (b) 6-nodes - Single Interface. (c) 8-nodes - Single Interface. (d) 12-nodes - Single Interface.



(e) 4-nodes - 4× Interfaces. (f) 6-nodes - 4× Interfaces. (g) 8-nodes - 4× Interfaces. (h) 12-nodes - 4× Interfaces.

Fig. 8: Execution time (in ms) for the *All-Gather* pattern for a message size of 100 MB.

TABLE IV: The performance of the *All-Reduce* pattern in terms of execution time, throughput, and number of hops.

Nodes	Placement Strategy	Message Size (in MB)	AI Collective	Execution Time (in ms)	Throughput (in Gb/s)	Throughput interf. (in Gb/s)	1-hop packets (in M)	3-hop packets (in M)
4	*4n_1r*	10.0	*All-Reduce*	**5.04 ± 0.00**	**399.82 ± 0.00**	**99.96 ± 0.00**	0.25 ± 0.00	–
	4n_2r			5.67 ± 0.28	360.61 ± 16.84	90.15 ± 4.21	0.08 ± 0.00	0.16 ± 0.00
	4n_3r			5.17 ± 0.14	390.82 ± 8.86	97.71 ± 2.21	0.04 ± 0.00	0.20 ± 0.00
	4n_4r			**5.04 ± 0.00**	**399.69 ± 0.00**	**99.92 ± 0.00**	–	0.25 ± 0.00
6	*6n_1r*	10.0	*All-Reduce*	**8.39 ± 0.00**	**599.83 ± 0.00**	**99.97 ± 0.00**	0.61 ± 0.00	–
	6n_2r			10.58 ± 0.45	481.79 ± 21.60	80.30 ± 3.60	0.25 ± 0.00	0.37 ± 0.00
	6n_3r			10.72 ± 0.46	476.42 ± 22.72	79.40 ± 3.79	0.12 ± 0.00	0.49 ± 0.00
	6n_4r			9.84 ± 0.41	517.88 ± 21.28	86.31 ± 3.55	0.08 ± 0.00	0.53 ± 0.00
8	*8n_2r*	10.0	*All-Reduce*	17.50 ± 0.42	539.13 ± 13.41	67.39 ± 1.68	0.49 ± 0.00	0.66 ± 0.00
	8n_3r			17.20 ± 0.50	549.73 ± 18.01	68.72 ± 2.25	0.29 ± 0.00	0.86 ± 0.00
	8n_4r			**16.58 ± 0.40**	**569.13 ± 14.93**	**71.14 ± 1.87**	0.16 ± 0.00	0.98 ± 0.00
12	*12n_2r*	10.0	*All-Reduce*	30.35 ± 0.36	730.41 ± 8.76	60.87 ± 0.73	1.23 ± 0.00	1.47 ± 0.00
	12n_3r			30.38 ± 0.36	729.66 ± 8.71	60.80 ± 0.73	0.74 ± 0.00	1.97 ± 0.00
	12n_4r			**29.92 ± 0.33**	**740.78 ± 8.15**	**61.73 ± 0.68**	0.49 ± 0.00	2.21 ± 0.00
4 & 6	*4n_1r_6n_1r*	10.0	*All-Reduce*	**8.89 ± 0.00**	**745.61 ± 0.00**	**74.56 ± 0.00**	0.81 ± 0.00	–
	4n_2r_6n_2r			**9.58 ± 0.26**	695.40 ± 19.01	69.54 ± 1.90	0.29 ± 0.00	0.52 ± 0.00
	4n_3r_6n_3r			10.38 ± 0.29	641.94 ± 18.22	64.19 ± 1.82	0.13 ± 0.00	0.68 ± 0.00
	4n_4r_6n_4r			10.40 ± 0.30	640.74 ± 17.93	64.07 ± 1.79	0.06 ± 0.00	0.75 ± 0.00
6 & 8	*6n_1r_8n_2r*	10.0	*All-Reduce*	**15.96 ± 0.40**	**870.67 ± 21.13**	**62.19 ± 1.51**	1.04 ± 0.00	0.65 ± 0.00
	6n_2r_8n_2r			**15.84 ± 0.39**	**877.66 ± 21.99**	**62.69 ± 1.57**	0.68 ± 0.00	1.01 ± 0.00
	6n_3r_8n_3r			17.18 ± 0.31	807.37 ± 14.07	57.67 ± 1.01	0.36 ± 0.00	1.33 ± 0.00
	6n_4r_8n_4r			17.31 ± 0.28	801.16 ± 13.11	57.23 ± 0.94	0.20 ± 0.00	1.48 ± 0.00

research will build on these findings by exploring additional DC infrastructures and a broader range of AI communication patterns. Furthermore, a novel scheduler will be designed and implemented to leverage these insights, further optimizing AI workload orchestration.

Acknowledgment

José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N. This work was performed during an internship at IBM Research, Yorktown Heights, NY, USA with financial support from the Research Foundation Flanders (FWO).

References

[1] D. Richins, D. Doshi, M. Blackmore, A. T. Nair, N. Pathapati, A. Patel, B. Daguman, D. Dobrijalowski, R. Illikkal, K. Long *et al.*, "Ai tax: The hidden cost of ai data center applications," *ACM Transactions on Computer Systems (TOCS)*, vol. 37, no. 1-4, pp. 1–32, 2021.

[2] J. Pan, L. Cai, S. Yan, and X. S. Shen, "Network for AI and AI for network: Challenges and opportunities for learning-oriented networks," *IEEE Network*, vol. 35, no. 6, pp. 270–277, 2021.

[3] S. Zeb, M. A. Rathore, S. A. Hassan, S. Raza, K. Dev, and G. Fortino, "Toward AI-enabled nextG networks with edge intelligence-assisted microservice orchestration," *IEEE Wireless Communications*, vol. 30, no. 3, pp. 148–156, 2023.

[4] J. Schneider, C. Meske, and P. Kuss, "Foundation models: a new paradigm for artificial intelligence," *Business & Information Systems Engineering*, pp. 1–11, 2024.

[5] L. Floridi, "AI as agency without intelligence: on ChatGPT, large language models, and other generative models," *Philosophy & technology*, vol. 36, no. 1, p. 15, 2023.

[6] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[7] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "{MegaScale}: Scaling large language model training to more than 10,000 {GPUs}," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760.

[8] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[9] P. Maniotis and D. M. Kuchta, "Exploring the benefits of using co-packaged optics in data center and AI supercomputer networks: a simulation-based analysis," *Journal of Optical Communications and Networking*, vol. 16, no. 2, pp. A143–A156, 2024.

[10] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346.

[11] C. Lai, Z. Zhou, A. Poptani, and W. Zhang, "LCM: LLM-focused Hybrid SPM-cache Architecture with Cache Management for Multi-Core AI Accelerators," in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, pp. 62–73.

[12] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang *et al.*, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 57–70.

[13] T. Gershon, S. Seelam, B. Belgodere, M. Bonilla, L. Hoang, D. Barnett, I. Chung, A. Mohan, M.-H. Chen, L. Luo *et al.*, "The infrastructure powering IBM's Gen AI model development," *arXiv preprint arXiv:2407.05467*, 2024.

[14] L. Schares, A. Tantawi, P. Maniotis, M.-H. Chen, C. Misale, S. Seelam, and H. Yu, "Chic-sched: a HPC Placement-Group Scheduler on Hierarchical Topologies with Constraints," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 424–434.

[15] ASTRA-sim, "An Open-source Distributed Deep Learning Training Simulator Infrastructure." accessed on 3 January 2025. [Online]. Available: https://astra-sim.github.io/.

[16] Z. Guo, S. Liu, and Z.-L. Zhang, "Traffic control for RDMA-enabled data center networks: A survey," *IEEE Systems Journal*, vol. 14, no. 1, pp. 677–688, 2019.

[17] P. Baziana, G. Drainakis, D. Georgantas, and A. Bogris, "AI and ML Applications Traffic: Designing Challenges for Performance Optimization of Optical Data Center Networks," in *2024 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2024, pp. 1–6.

[18] J. Hu, H. Shen, X. Liu, and J. Wang, "RDMA Transports in Datacenter Networks: Survey," *IEEE Network*, 2024.

[19] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, and L. Cui, "Congestion control for cross-datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 30, no. 5, pp. 2074–2089, 2022.

[20] V. Addanki, O. Michel, and S. Schmid, "{PowerTCP}: Pushing the performance limits of datacenter networks," in *19th USENIX symposium on networked systems design and implementation (NSDI 22)*, 2022, pp. 51–70.

[21] B. Nougnanke, Y. Labit, M. Bruyere, U. Aivodji, and S. Ferlin, "ML-based performance modeling in SDN-enabled data center networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 815–829, 2022.

[22] T. A. Khan, M. S. Khan, S. Abbas, J. I. Janjua, S. S. Muhammad, and M. Asif, "Topology-Aware Load Balancing in Datacenter Networks," in *2021 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*. IEEE, 2021, pp. 220–225.

[23] P. Cao, S. Zhao, D. Zhang, Z. Liu, M. Xu, M. Y. Teh, Y. Liu, X. Wang, and C. Zhou, "Threshold-based routing-topology co-design for optical data center," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 2870–2885, 2023.

[24] C. Gonzalez and B. Tang, "AggVNF: Aggregate VNF Allocation and Migration in Dynamic Cloud Data Centers," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE, 2024, pp. 73–81.

[25] X. Guo, X. Xue, F. Yan, B. Pan, G. Exarchakos, and N. Calabretta, "DACON: a reconfigurable application-centric optical network for disaggregated data center infrastructures," *Journal of Optical Communications and Networking*, vol. 14, no. 1, pp. A69–A80, 2022.

[26] A. S. Alhazmi, S. H. Mohamed, T. E. El-Gorashi, and J. M. Elmirghani, "OWC-enabled Spine and Leaf Architecture Towards Energy Efficient Data Center Networks," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. IEEE, 2022, pp. 13–18.

[27] J. Yoo, W. Won, M. Cowan, N. Jiang, B. Klenk, S. Sridharan, and T. Krishna, "Towards a Standardized Representation for Deep Learning Collective Algorithms," in *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2024, pp. 33–36.

[28] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "Astra-sim2. 0: Modeling hierarchical networks and disaggregated systems for large-model training at scale," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2023, pp. 283–294.

[29] J. Cho, M. Kim, H. Choi, and J. Park, "LLM-Sim: A Simulation Infrastructure for LLM Inference Serving Systems," in *Machine Learning for Computer Architecture and Systems 2024*, 2024.

[30] D. Van Poucke, W. Tavernier, and D. Colle, "Network-Centered Resource Management for HPC Networks," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE, 2024, pp. 235–238.

[31] L. Campanile, M. Gribaudo, M. Iacono, F. Marulli, and M. Mastroianni, "Computer network simulation with ns-3: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 272, 2020.

[32] A. Varga, "A practical introduction to the OMNeT++ simulation framework," in *Recent advances in network simulation: the OMNeT++ environment and its ecosystem*. Springer, 2019, pp. 3–51.

[33] R. Birke, G. Rodriguez, C. Minkenberg *et al.*, "Towards massively parallel simulations of massively parallel high-performance computing systems." in *SimuTools*, 2012, pp. 291–298.

[34] A. Varga, "Discrete event simulation system," in *Proc. of the European Simulation Multiconference (ESM'2001)*, vol. 17, 2001.

[35] P. Maniotis, L. Schares, B. G. Lee, M. A. Taubenblatt, and D. M. Kuchta, "Scaling HPC networks with co-packaged optics," in *Optical Fiber Communication Conference*. Optica Publishing Group, 2020, pp. T3K–7.

[36] ——, "Toward lower-diameter large-scale HPC and data center networks with co-packaged optics," *Journal of Optical Communications and Networking*, vol. 13, no. 1, pp. A67–A77, 2021.

[37] Q. Zhou, Q. Anthony, L. Xu, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. D. Panda, "Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 134–144.

[38] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[39] IBM Research, "Why we built an AI supercomputer in the cloud." accessed on 3 January 2025. [Online]. Available: https://research.ibm.com/blog/AI-supercomputer-Vela-GPU-cluster.