Vela: A Virtualized LLM Training System With GPU Direct RoCE

Apoorve Mohan*
IBM Research
Yorktown Heights, USA

Robert Walkup
IBM Research
Yorktown Heights, USA

Bengi Karacali
IBM Research
Yorktown Heights, USA

Ming-hung Chen
IBM Research
Yorktown Heights, USA

Abdullah Kayi
IBM Research
Bethesda, USA

Liran Schour
IBM Research
Haifa, Israel

Shweta Salaria
IBM Research
Yorktown Heights, USA

Sophia Wen
IBM Research
Yorktown Heights, USA

I-hsin Chung
IBM Research
Yorktown Heights, USA

Abdul Alim
IBM Research
Yorktown Heights, USA

Constantinos Evangelinos
IBM Research
Cambridge, USA

Lixiang Luo
IBM Research
Yorktown Heights, USA

Marc Dombrowa
IBM Research
Yorktown Heights, USA

Laurent Schares
IBM Research
Yorktown Heights, USA

Ali Sydney
IBM Research
Cambridge, USA

Pavlos Maniotis
IBM Research
Yorktown Heights, USA

Sandhya Koteshwara
IBM Research
Yorktown Heights, USA

Brent Tang
IBM Cloud
Rochester, USA

Joel Belog
IBM Cloud
Lowell, USA

Rei Odaira
IBM Cloud
Austin, USA

Vasily Tarasov
IBM Research
Almaden, USA

Eran Gampel
IBM Cloud
Haifa, Israel

Drew Thorstensen
IBM Cloud
Durham, USA

Talia Gershon
IBM Research
Yorktown Heights, USA

Seetharami Seelam*
IBM Research
Yorktown Heights, USA

Abstract

Vela is a cloud-native system designed for LLM training workloads built using off-the-shelf hardware, Linux KVM-based virtualization, and a virtualized RDMA over Converged Ethernet (RoCE) network. Vela virtual machines (VMs) support peer-to-peer DMA between the GPUs and SRIOV-based network interface.

In this paper, we share Vela's key architectural aspects with details from an NVIDIA A100 GPU-based deployment in one of the IBM Cloud data centers. Throughout the paper, we share insights and experiences from designing, building, and operating the system over a ~2.5 year timeframe to highlight the capabilities of readily available software and hardware technologies and the improvement opportunities for future AI systems, thereby making AI infrastructure more accessible to a broader community. As we evaluated the system for performance at ~1500 GPU scale, we achieved ~80% of the ideal throughput while training a 50 billion parameter decoder model using model parallelism, and ~70%

*Corresponding Authors: apoorve.mohan@ibm.com, sseelam@us.ibm.com

ASPLOS '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1079-7/2025/03

per GPU FLOPS compared to a single VM with the High-Performance Linpack benchmark.

CCS Concepts: • **Software and its engineering → Virtual machines**; *Operating systems*; *Massively parallel systems*; *Distributed systems organizing principles*; • **Computer systems organization → Cloud computing**; • **Networks** → *Network architectures*; *Network protocols.*

Keywords: Virtualization, Cloud Computing, High Performance Computing, AI Infrastructure

ACM Reference Format:
Apoorve Mohan, Robert Walkup, Bengi Karacali, Ming-hung Chen, Abdullah Kayi, Liran Schour, Shweta Salaria, Sophia Wen, I-hsin Chung, Abdul Alim, Constantinos Evangelinos, Lixiang Luo, Marc Dombrowa, Laurent Schares, Ali Sydney, Pavlos Maniotis, Sandhya Koteshwara, Brent Tang, Joel Belog, Rei Odaira, Vasily Tarasov, Eran Gampel, Drew Thorstensen, Talia Gershon, and Seetharami Seelam. 2025. Vela: A Virtualized LLM Training System With GPU Direct RoCE. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, pages.

1 Introduction

Large Language Model (LLM) training workloads and high-performance computing (HPC) workloads share several peculiar characteristics [, ,]. For example, HPC workloads such as simulations and mathematical modeling require handling parallel processing of large datasets and matrix multiplication operations. Similar computational and data processing patterns also constitute the core of the training process of neural networks for LLMs. Thus, like HPC workloads, LLM training workloads are computationally intensive as they process large datasets, perform complex calculations, and benefit from parallel and distributed processing technologies. Leveraging parallel and distributed computing paradigms can be crucial to optimizing performance and reducing computation time when processing complex computations, as required for LLM training. Complex calculations are broken into smaller tasks across multiple computation units within and across nodes and periodically synchronized over an intra and inter-node network, respectively.

Traditionally, organizations have deployed on-premise bare-metal clusters to support HPC-style workloads. Such clusters typically consist of special-purpose servers/nodes for storage and compute requirements connected via a high-throughput, low-latency network fabric built and optimized for workload-specific collective communications and minimizing network congestion. The hardware and software components that drive such clusters are often performance-centric towards specific workloads. While these systems provide the best in class performance, they are infamous for being limited in system flexibility and maintainability – which the cloud computing paradigm aims to overcome.

Cloud computing started by offering virtual machine (VM) based solutions for general-purpose infrastructure. However, due to the growing demand to support resource-intensive workloads such as LLM training, most hyperscalers today also offer VM-based HPC-style GPU-based Infrastructure-as-a-Service (GPUaaS) [,]. While using these services, customers can customize the software (including the OS) and runtime (e.g., TensorFlow, PyTorch, CUDA) to meet their specific requirements. Like general-purpose infrastructure offerings, GPUaaS offerings are scalable, where the infrastructure can be (re-)provisioned for different customers multiple times daily. Provisioning infrastructure, managing failures, and maintenance activities also become easy for administrators with the availability of features like VM migration driven by well-defined APIs, i.e., Infrastructure-as-Code [ ].

In our quest to build a cloud-native system to support LLM training workloads, we explored the existing cloud system architectures and how they support HPC-style workloads, and one vital aspect stood out. Due to their sheer size (i.e., economies of scale), cloud hyperscalers invest in system design decisions that might not be economically feasible for most organizations (e.g., academic institutions, medical research laboratories) interested in building cloud-native systems to support HPC-style workloads. Examples of such decisions include building specialized hardware [, , , ,], deploying supercomputer-style networks [ ], or developing custom virtualization stacks [,]. So the challenge for us was to answer the question: *"Can we build a cloud (virtualization-based) system for LLM training workloads with off-the-shelf hardware and a primarily open-source system stack while delivering HPC-like performance?"*.

Our Approach. To answer the above question, we built Vela. Vela is a cloud-native system designed for LLM training workloads built using off-the-shelf hardware, Linux KVM-based virtualization, and a virtualized RDMA over Converged Ethernet (RoCE) network. Vela leverages PCIe device passthrough to achieve near-bare-metal performance for workloads within the Linux-KVM-based virtual machines (VM). To enable low latency, high throughput, and fault-tolerant communication between the VMs, a vendor-agnostic flow-based software-defined networking (SDN) solution is employed in Vela. The SDN solution leverages Single Root I/O Virtualization (SRIOV) technology to create virtual functions (VFs) on top of network interface cards (NICs), which are passthrough to the VMs. The hypervisor and the SDN solution work together to enable GPU Direct RDMA over Converged Ethernet (GDR) based networking between the VMs. Vela's RoCE-based network is designed to offer a lossless experience over a lossy network through a vendor-agnostic hardware-software co-design approach for congestion management (leveraging techniques such as Equal Cost Multi-pathing (ECMP), Network Flow Control, and RDMA Queue

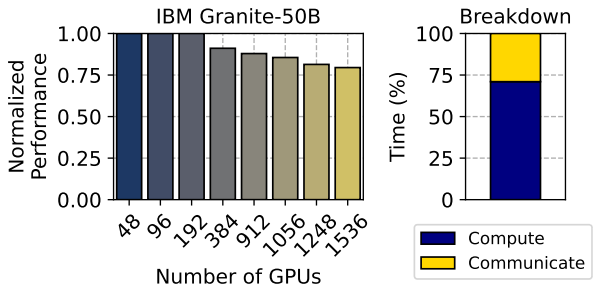


Figure 1. Training performance of IBM Granite-50B model on Vela at different scales normalized to ideal throughput.

Pairs (QPs)). Vela employs a three-tier storage hierarchy to prevent bottlenecks while accessing data during the different stages of the LLM training workflow.

Contributions. The contributions of this paper are as follows: (i) A summary of the key architectural aspects of Vela's compute, network, and storage components. (ii) Scaling performance evaluation and details from an NVIDIA A100 GPU-based Vela deployment located in one of our data centers. (iii) A case-study-based discussion on the importance of the network for LLM training workloads for Pytorch Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP) strategies. (iv) Experiences from building, deploying, and operating the aforementioned Vela deployment over a ~2.5 year period, including critical functionality, security, and performance issues. (v) Details on setting up the Linux-KVM-based system stack required to achieve near-bare-metal performance for LLM training workloads . (vi) An open-source Pytorch-based benchmark suite for the commonly used collective communication operations employed in LLM training workloads[1]. (vii) Code to replicate hypervisor crash from a VM by fuzzing passthrough devices[1].

2 Vela Overview

Vela is a horizontally scalable cloud-native system designed for LLM training workloads. Figure (network) shows the system/rack-level view of Vela, and Figure (compute) shows the architecture of the GPU nodes in Vela. In this section we discuss the key architectural aspects, and at-scale network and application performance achieved on a Vela deployment from one of our data centers.

2.1 Virtualization

Compute. Vela presents compute nodes as KVM+QEMU virtual machines (VMs) with the GPUs, the highspeed Intra-Node fabric connecting the GPUs, and the network SRIOV-VFs in passthrough mode, and we construct the VMs such that they preserve the PCIe topology and NUMAness of the underlying system. With the PCIe device passthrough technique, the guest OS can directly access the physical devices installed on the host system in a performant and IOMMU-protected manner if the underlying hypervisor's kernel has support for the Virtual Function I/O (VFIO) driver. As a result, applications running inside VMs can take advantage of the full power of devices like GPUs and enable performant execution of applications. We employ large memory and storage configurations to support use cases, such as caching AI training data, models, and other related artifacts and feeding the GPUs with data to keep them busy.

GPU Direct RoCE (GDR): One of the critical aspects of Vela that enabled high-throughput, low-latency communication between the VMs was the enablement of Peer-to-Peer Direct Memory Access (P2P DMA) between the passthrough GPUs and RDMA-capable network SRIOV-VFs from inside the VMs Guest OS. The Peripheral Component Interconnect Express (PCIe) standards allow P2P DMA between two PCIe-capable devices to exchange data without involving the CPU. This capability is the fundamental basis of GPU Direct technologies implemented by various accelerator vendors [,], which enables DMA transfers between a GPU and other DMA-capable devices (such as RDMA-capable NICs and NVMe drives). The role of the CPU in the GPU Direct workflow is minimal, i.e., restricted to initialization/orchestration of P2P DMA between the devices, and the main memory is not involved in data transfers as the devices can directly read from or write to each other's memory (i.e., additional data copies between between the devices and main memory is prevented). For example, during the communication phase of a training workload when a collective operation (e.g., AllReduce) is performed, the data structure in the GPU memory will be DMA'd to the NIC without the need for it to be transferred to the CPU memory and can significantly improve the aggregate throughput of collective calls as discussed in Section . Note that Vela's RDMA implementation is based on RDMA over converged Ethernet (RoCE) technology.

Enabling GDR in VMs: In a virtualized environment, PCIe devices are often subsetted into separate security domains, i.e., IOMMU groups, to prevent unauthorized access. This is a major challenge in enabling GDR in VMs, as the PCIe Access Control Services (ACS) forces the PCIe traffic to be routed through the PCIe root complex to avoid the violation of IOMMU groupings, disabling P2P DMA between devices. As a workaround to this security requirement, PCIe Address Translation Services (ATS) allows caching address translations at PCIe device endpoints after initial verification [ ]. Vela employs this technique to enable P2P DMA between the GPU and network SRIOV-VFs. Enabling a performant VM-based execution environment for AI workloads required changes at different system layers: (a) The system

[1]

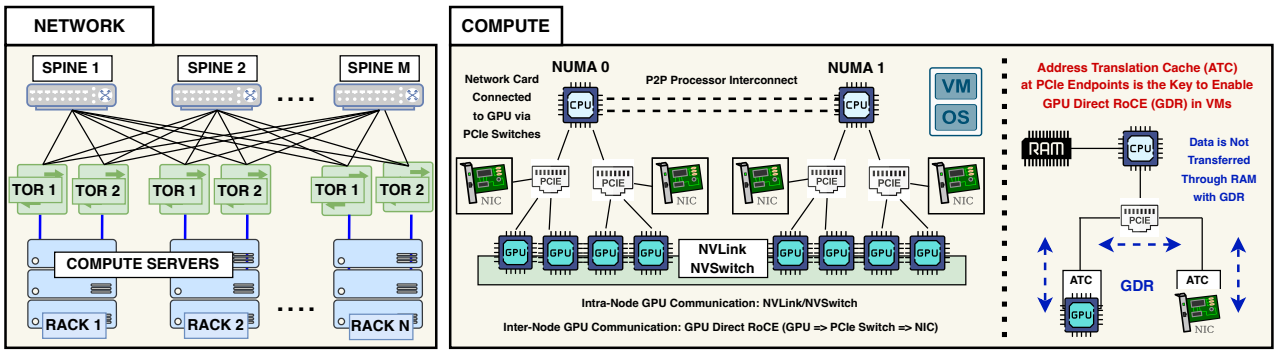


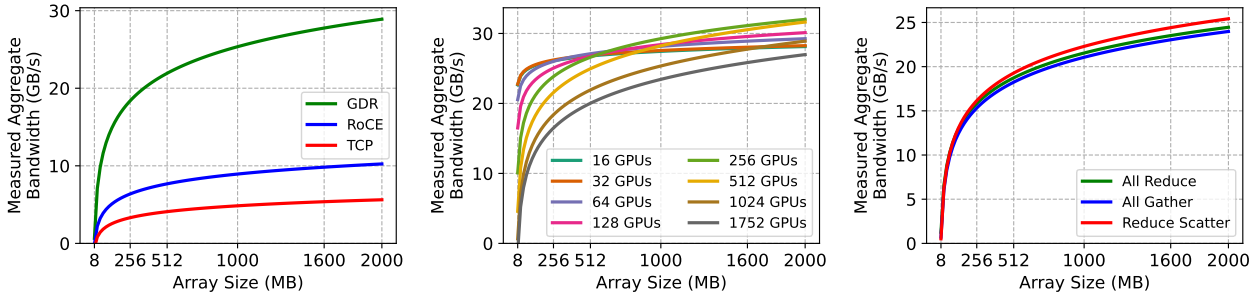
Figure 2. Vela: Compute and Network Architecture

BIOS includes enabling hardware-assisted virtualization support (e.g., VT-d, VMX, and IOMMU on an Intel-based x86 platform), PCIe Access Control Services (ACS), and Single Root I/O Virtualization support; (b) In the network adapter this includes enabling relaxed PCIe ordering, increasing maximum accumulative outstanding read bytes, enabling selective repeat, enabling address translation caching for peer-to-peer DMA performance, and enabling direct access to ATS from the NIC to GPU buffers using PCIe peer-to-peer transactions; (c) In the hypervisor, enable network SRIOV-VFs, huge pages, ACS on the PCI controllers, ATS on the network SRIOV-VFs, and increase the maximum PCI read request size to 4KB; (d) In the guest domain definition (typically in XML format), enable huge pages, NUMA domains, device-NUMA mapping, host-physical-bit model for large memory VMs, and ATS on PCI controllers; and (e) In the guest OS (increase maximum PCI read request size to 4KB).

Network. Vela employs a vendor-agnostic flow-based software-defined network (SDN) solution on the compute nodes to enable secure and performant communication between virtual machines. The SDN solution ensures isolation to support multi-tenancy, supports unpredictable tenant traffic patterns and maximizes available bandwidth and performance for tenant traffic. The goal of the SDN solution is three-fold: (a) to provide an endpoint-centric solution where the virtual network processing happens on the compute nodes instead of a middle box; (b) to accelerate packet processing via hardware offloading; and (c) to enable guests with the freedom to select a virtual interface type for their VM instances. We employed a "bump in the wire" hardware approach to realize endpoint-centric processing. In conjunction, we use software pipelining as the slow path while directing most traffic to the NIC hardware pipeline as the fast path.

Our design principle of hardware offloading requires processing power at the endpoints to execute network processing on each packet. To minimize computation cycles on the hosts, we utilize programmable NICs to offload network virtualization processing to the NIC hardware. This approach reduces host CPU consumption and provides lower latency, low jitter, and high throughput. Note that our approach is not limited to a specific vendor, as we were able to integrate multiple vendors and in-house technologies. Throughout our efforts, we encountered changes in NIC hardware capabilities and emerging technologies, such as P4-based NIC architecture [,]. While adhering to our principle of offloading computation to the NIC, we adapted to the varying capabilities of each NIC vendor used in our SDN solution. An essential capability we needed to adapt to was the supported NIC hardware rule scale, which refers to the maximum number of hardware rules and the hardware update rate supported by the NIC. Initially, we used wildcard hardware rules matching to reduce the number of rules, avoiding a one-to-one relationship for session-to-hardware rule pairs. However, we discovered that wildcard-based hardware pipeline matching led to performance degradation compared to exact-match hardware rules for most vendors. *Note:* In the wildcard approach *"one"* hardware rule maps to many active flows, whereas, in exact match approach there is a *"one-to-one"* mapping between the number of hardware rules and active flows. We observe that for AI training workloads, we have low number of active flows as compared to large number of active flows observed in traditional cloud workload [ ].

Software Pipeline: Vela's software pipeline is a Data Plane Development Kit (DPDK) based application designed to implement a comprehensive set of network functionalities commonly used in cloud environments. It is implemented in a modular design, featuring a packet processing graph composed of various modules. Each module is responsible for a specific functionality within the pipeline: e.g., logical connectivity, firewall, load balancing, encapsulation, hardware offloading. Our design strives to achieve an endpoint-centric processing approach eliminating middle boxes and pushing virtual network processing to the endpoints SDN pipeline. An example of this approach is implementing virtual router functionality in a distributed manner (known as distributed virtual routing (DVR)) at the endpoint, eliminating the need for a single component in the system to function as the



(a) 1024 GPUs AllReduce performance. **(b)** GDR scaling performance. **(c)** 1280 GPUs collective performance.

Figure 3. At-scale network collective call performance.

virtual router. This distributed pattern of network virtualization, extending to functionalities such as virtual routers, network address translation (NAT), and load balancing (LB), is a widely adopted industry practice. It facilitates scalability and high performance without a single point of failure.

Our pipeline based approach is vendor agnostic design. Regardless of the NIC hardware deployed, our solution aims to optimize hardware offloading resources while maintaining a unified SDN stack solution agnostic to the NIC hardware. We developed a generic flow-based abstraction to provide a unified interface to the SDN stack, maximizing hardware offloading capabilities regardless of the specific NIC vendor, technology, or supported API. Supported APIs could include TC Flower, RTE Flow, P4 Portable NIC Architecture (PNA), or any other vendor-specific API, such as emerging P4-based NICs pipelines/SDKs. Our exploration revealed that flow-based exact match can accommodate all possible use cases in the cloud while delivering advanced hardware offloading performance. It's adaptable to all hardware vendors. Initially, we utilized wildcards in the hardware pipeline design to reduce the total number of required hardware rules. Transitioning to a flow-based approach, we developed a mechanism enabling flexible decisions during hardware offloading based on NIC capabilities, flow use cases (e.g., east-west vs. north-south NAT), and flow characteristics (e.g., mice vs. elephants flows).

Given the requirement to maintain a multi-vendor solution and intelligent offloading decision-making, we opted for the offline approach, managing hardware offloading with a dedicated worker. We devised a configuration for offloading criteria analyzed in runtime while processing packets using an offload predictor module, which lets us determine the flows we can offload to the hardware. Initially, we implemented simple criteria based on the session lifetime or the packet count within a time threshold (PPS). For instance, we optimized the system to offload all RDMA AI training workloads (e.g., RoCE packets) without delay, configuring high QoS class types to achieve ultra-low latency. We avoid offloading to hardware for short-lived mice flows and instead continue processing in software. Subsequently, we added

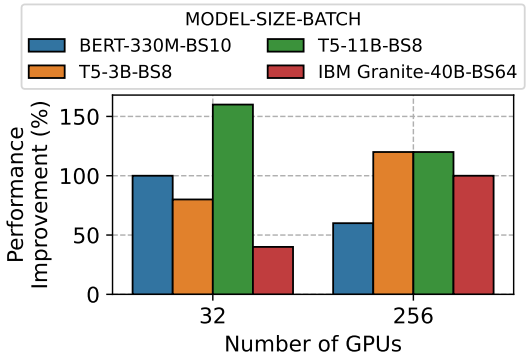


Figure 4. Model Performance: TCP vs GDR

more advanced criteria based on learned customer traffic patterns for each traffic use case.

2.2 Communication

Vela employs a two-level spine-leaf Clos topology for network connectivity (as shown in Figure (network)) and uses Ethernet-based networking to support Transmission Control Protocol (TCP) and RDMA over Converged Ethernet (RoCE) based communication. The training workloads primarily use GPU Direct + RoCE (GDR) for inter-node communication. Vela's RoCE deployment strives to offer a lossless experience over a lossy network and employs Equal-Cost Multipath (ECMP) routing. Achieving high performance and managing entropy was critical to achieve scaling efficiency for the RoCE network and required optimizations, such as deploying a QoS profile on the switch, tuning of Explicit Congestion Notification (ECN) based congestion control, tuning the switching buffers, applying properties such as flow characteristics, and configuring optimal RDMA queue pairs (QPs) at the application layer..

For congestion control, we isolate RoCE traffic in a traffic class, monitor the congestion experienced by this class in the network, and notify the senders to reduce their traffic to alleviate congestion before packet losses occur. This process works by first marking the type of service (ToS) byte in the

header of RoCE packets at the source. The first six bits of the ToS byte are reserved for the Differentiated Services Code Point (DSCP) tag, and the last two bits are reserved for the Explicit Congestion Notification (ECN) tag. RoCE traffic is marked with a specific DSCP tag and an ECN value to enable the switches to process RoCE traffic via a dedicated queue and to indicate the occurrence of congestion in this queue in the ECN field, respectively.

Further, we configure the network switches to monitor the buildup in the RoCE queue as they determine if there is congestion based on the length of the RoCE queue and mark the ECN field accordingly. Upon receiving a RoCE packet with a congestion indication, the receiver sends a high-priority message to the sender, the Congestion Notification Packet (CNP). Upon receiving a CNP packet, the sender throttles its traffic injection rate to reduce congestion. The network switches are configured to route high-priority packets and are tuned to minimize packet losses. Implementing congestion control also required changes on compute nodes, including configuring the network interface cards (NICs) to enable congestion control mechanisms, enabling the SDN to mark RoCE traffic with a specific DSCP tag (which the network would recognize switches for traffic isolation), and support ECN. To support high availability, we also built network redundancy into the system. Each port of the network interface card (NIC) is connected to a different top-of-rack (TOR) switch (as shown in Figure (network)). *Note:* We did not modify the DCQCN mechanism but optimized/tuned parameters for our environment to achieve optimal performance for the training workloads.

In RoCE networks, entropy is a critical component as it determines how traffic is distributed across the network paths to help ensure balanced load distribution and minimize congestion and latency. In Vela ECMP hashing and RDMA Queue Pairs work in tandem to manage entropy and optimize network performance. ECMP hashing uses the header fields of packets to decide which path to take. Each RDMA flow is mapped to one of the multiple equal-cost paths based on the hash result, and each QP may be directed to a different network path, which will help distribute the load across the fabric. An application can use multiple QPs to achieve higher entropy. By spreading traffic across QPs, the network can generate more diverse hash values, leading to a more even distribution across paths. In large-scale data centers, where individual flows may be heavy, using multiple QPs can help avoid overloading a single path.

2.3 Data Access

During a large-scale model training job, several data- and I/O-intensive steps can become a bottleneck to the overall training job progression without an optimized storage solution. The first occurrence is when the GPUs access data stored in the cloud object store (COS) to begin training. Loading data directly from object storage to each GPU's memory is slow due to the limited performance (e.g., Input/Output Operations per second) supported by typical cloud object storage. This bottleneck occurs at the beginning of the training job, and every time, the job stops and needs to be re-started. System component failures make starting and stopping of jobs inevitable. A second I/O-intensive step of a model training job occurs during model "checkpointing." At periodic intervals during training, a record of the current set of model weights is sent to object storage, similar to occasionally "saving" the collective work of the GPUs. In both examples, a high-performance file system between the object storage and the GPUs can act as an intermediate caching solution. When using a high-performance file system, data can be loaded into the GPUs much faster to start (or re-start) a training job, and model weights can be checkpointed to the file system at a much faster rate than to the object storage.

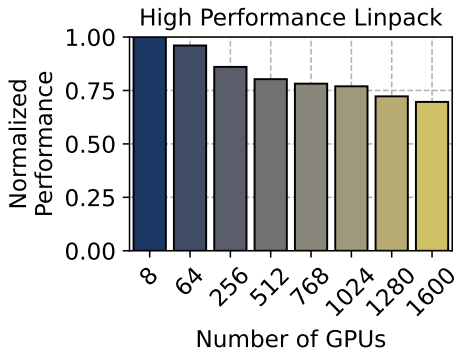


Figure 5. Per GPU FLOPS compared to a single VM.

Vela employs a three-tier hierarchical storage for LLM training workloads. The first layer consists of high-speed node-local drives to be used by workloads for storing transient data such as intermittent checkpoints and cache. The second layer is a high-performance shared filesystem for I/O intensive multi-node workloads for caching large data sets and sharing or synchronizing data during workload execution. The third and final layer is a data lake to store long-term write-once read-only data. In the case of Vela, layer one uses high-speed local NVMe drives; for layer two, a General Parallel File System (GPFS) is used; and layer three is implemented using a COS. The GPFS storage cluster is deployed using a disaggregated storage model on top of tens of VMs with two 1TB virtual block volumes attached to each instance. The virtual block volumes are hosted on a next-generation cloud-native and highly performant block storage service that can meet the high throughput requirements of model training workloads. A single file system is created using all attached devices, enabling hundreds of Terabytes to be the total capacity of the GPFS (can be extended to PBs).

For cost efficiency, most of the data used by the training jobs originates from the COS – even the checkpoint data produced by the jobs (to be used for job restarts) must end up in

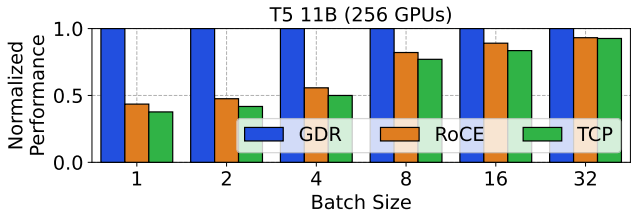


Figure 6. T5 11B training performance.

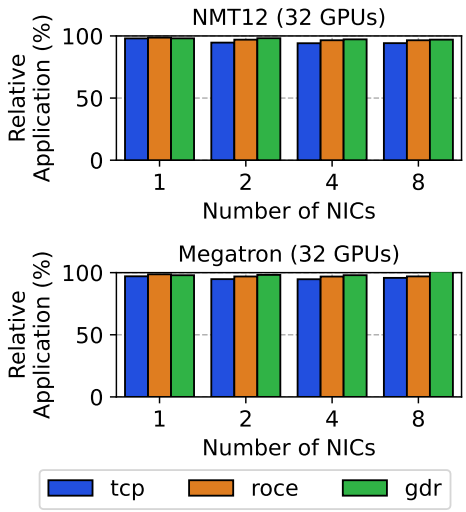


Figure 7. Multi-VM Performance.

the COS. However, accessing COS is slow and can quickly become a bottleneck – which is resolved using GPFS in the case of Vela. The file sets on the GPFS are transparently connected to the object storage buckets. File system namespaces represent objects in buckets as files and brings data from the object storage into the file system on demand. In doing so, the GPFS file system acts as a read-write cache over potential petabytes of cost-effective object storage. When the cache is at capacity, the least recently used data and metadata are automatically evicted to the COS.

2.4 Deployment

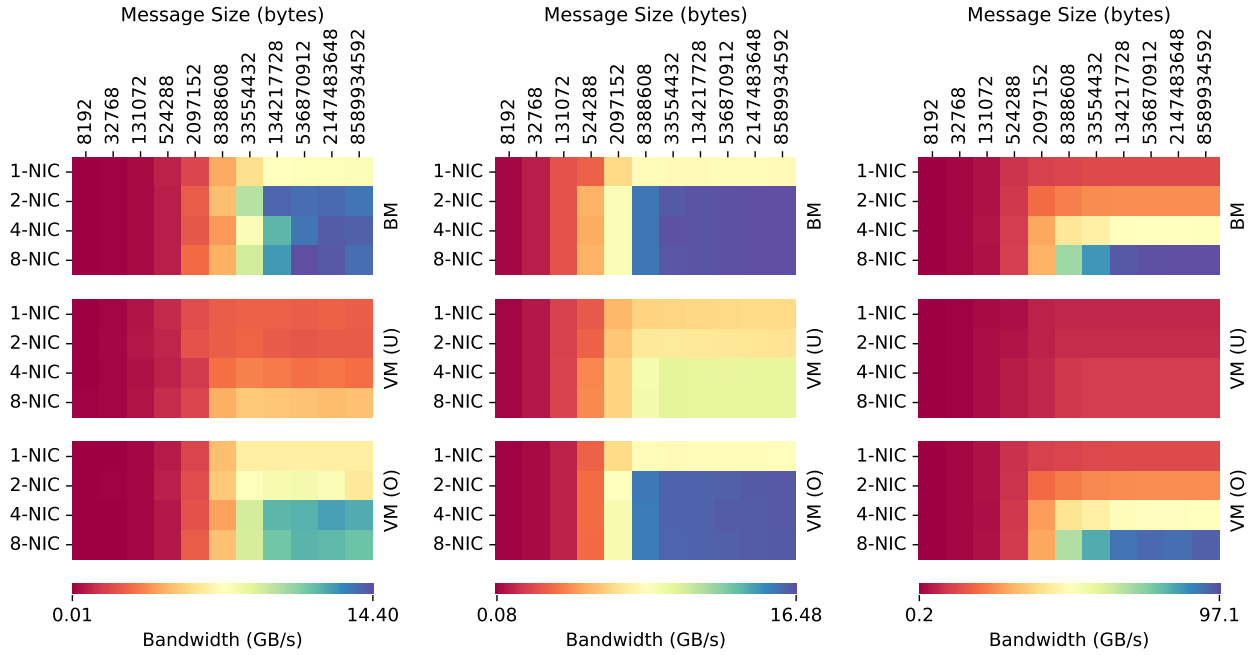
The at-scale performance results and experiences presented in the following sections are based on a two-phase deployment and usage over a ~2.5 year period. The first phase enabled performant execution of GPU workloads inside VMs with TCP-based inter-VM communication with a maximum of three nodes per rack. Each VM runs on top of an Intel Cascade Lake-based system with 80 vCPUs, 1.25 TB of vRAM, 4x3.2TB NVMe, and 8xNvidia A100 80GB GPUs connected via NVLINK/NVSwitch. In addition, VMs have 2xSRIOV VFs, where each SRIOV-VF runs on top of a ConnectX Dual Port 2x100Gbps NIC . Each physical NIC port connects to a different top-of-rack (TOR) switch, and each TOR switch is connected via two 100G links to four spine switches, providing 1.6Tbps cross-rack bidirectional bandwidth and ensuring that the system can continue to operate despite failures of any given NIC, TOR, or spine switch. The connectivity between the CPU and the PCIe switch (as shown in Fig) is PCIe Gen3-based, and the connectivity from the PCIe switch to the GPUs is PCIe Gen4-based. The second phase enabled performant GDR-based inter-VM communication and increased the cluster's compute capacity by 2X to a maximum of six nodes per rack. Note that we did not increase the network capacity in phase two and over-committed the spine switches by 50%. In the first phase, NFS was used as an intermediate file system between the compute nodes and the COS, which was replaced by a high-performance GPFS solution.

Power. Each Vela rack has six servers instead of the industry norm of three/four servers. Typical cloud racks are rated to provide 20kW of power to each rack from each of two redundant power distribution units (PDUs) (for a total of 40 kW available). Each GPU server draws a maximum of 6kW of power. Therefore, three servers can be accommodated per rack while preserving full power redundancy. We notice that systems can accommodate greater density if a mechanism is in place to address potential failures of a power supply unit (PSU) – where the servers are automatically throttled down to avoid overloading the healthy PDU. With the help from vendors we enabled an optimized power brake solution to enable power overcommitment. When a PSU fails, the updated server firmware applies the power brake solution in about 2 seconds, and the system throttles down to 3.2kW (taking each GPU from roughly 400W to 150W). Healthy PDU circuit breakers can tolerate power surges of up to 5 seconds – thus enabling safe overcommitment of the amount of power available to each rack.

2.5 Scaling Performance

Network. Distributed workloads (e.g., LLM training) that require periodic communication between different workers tend to become communication-bound as they are horizontally scaled out, and the systems network performance can become a gating factor for them and lead to performance inefficiencies. Thus, we wanted to understand how Vela's network performs at scale after completing the second deployment phase. We start by measuring the AllReduce collective operation performance with TCP, RoCE, and GDR communication protocols for different array sizes at a 1024 GPU scale (i.e., 128 VMs). As seen in Figure , the peak performance of the AllReduce collective operation with GDR outperforms TCP and RoCE by a factor of ~5x and ~2x, respectively. The difference between TCP and GDR appears

[2]Note that for the virtualization overhead experiments, we enabled 8x100Gbps network SRIOV-VFs on a 4 nodes (i.e., 32 GPUs) testbed.



(a) TCP **(b)** RoCE **(c)** GDR

Figure 8. NCCL AllReduce Performance: Bare Metal (BM), Unoptimized VM (VM-U), and Optimized VM (VM-O).

to be significant with this measurement; however, for real LLM training workloads, we observe ~2x the improvements with GDR (see Figure). Further, Section presents a case study on network capacity's (un-)importance on LLM training workloads. Note that with Vela's Ethernet-based virtualized network (overprovisioned at the spine layer after the second phase deployment), at the 1024 GPU scale, the measured peak aggregate bandwidth for the AllReduce collective operation with GDR was ~70% of the theoretical hardware limit.

As we scaled the AllReduce operation from 16 to 1752 GPUs using GDR, we observed a significant decline in performance for smaller array sizes (see Figure). For instance, at 1752 GPUs, the AllReduce operation on a 64MB array experienced a ~78% degradation compared to its performance at 16 GPUs. This decline suggests large-scale LLM training can become latency-bound when exchanging smaller gradient updates. Recent large-scale LLM training parallelization strategies such as Fully Sharded Data Parallelism [ ] decompose AllReduce collective operations into ReduceScatter and AllGather collective operations to shard parameters and gradients. Thus, we also compared the performance of AllReduce against ReduceScatter and AllGather at a 1280-GPU scale and found no significant difference (see Figure).

Application. To demonstrate the application scaling performance of Vela, we use the High Performance Linpack (HPL) benchmark and IBM Granite-50B model training. HPL is a well-known benchmark used for measuring the efficiency of Supercomputers and HPC systems [ ]. Figure shows the per GPU performance as we scale from 1 VM (i.e., 8 GPUs) to 200 VMs (1600 GPUs). In the figure, we report the percentage per GPU FLOPS for each scale relative to what we measure on a single VM (i.e., 8 GPUs), which shows that at a scale of 512 GPUs and ~1500 GPUs, this Vela deployment achieves ~80% and ~70% per GPU FLOPS compared to a single VM respectively. Next, we discuss the performance of LLM training. First, we show how a IBM Granite-50B model performs at different scales when trained using model parallelism. Figure shows the measured throughput relative to the ideal expected system throughput and the breakdown of the percentage time spent in computation vs communication for the said LLM training job. In our measurement, we observed that the job scaled near perfectly for up to ~256 GPU scale. As we scaled the training job to ~1500 GPUs, we achieved ~80% of the ideal system throughput, and we were able to achieve ~45% Model FLOPs Utilization (MFU).

2.6 Discussion: Importance of Network?

Distributed Data Parallel (DDP). In the PyTorch DDP method, model parameters are replicated across GPUs, and each GPU processes a subset of data. The gradients of the parameters are periodically summed across all GPUs using a collective communication operation, AllReduce(), during the back-propagation phase. PyTorch DDP attempts to overlap this communication with GPU computation as much as possible by making calls to AllReduce() when enough gradients

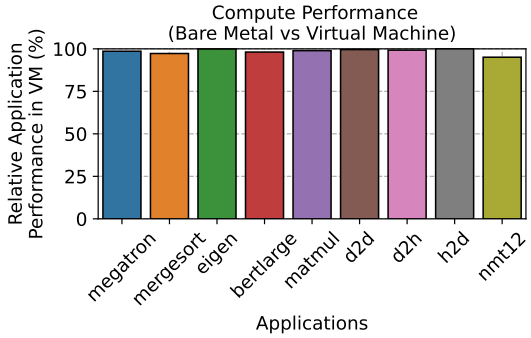


Figure 9. Single-VM Performance.

have been computed to fill a "bucket". In principle reductions can proceed once gradients for a given model layer have been computed. PyTorch DDP allows the user to specify a bucket size when the method is instantiated. For relatively small AI models like BERT-Large, selecting a bucket size larger than the default (25 MB) can be beneficial. BERT-Large has 24 model layers and ~340M parameters, so when gradients are in FP32 format, the gradient array size is ~56.7MB per layer. By specifying a bucket size of ~200 MB, reductions should occur every four layers instead of layer by layer. Reductions with the larger arrays are less sensitive to network latency, and can provide improved scaling.

We investigated the scaling of BERT-Large pre-training on our GPU cluster using PyTorch DDP. We used the training code outlined in [ ] with a local batch size of 40 per GPU. When training on a single node with eight NVIDIA A100 GPUs connected by NVLink, all communication is by peer-to-peer transfers between GPUs via NVLink, which provides very high bandwidth and low latency. Using the Pytorch profiler, we observed that all calls except the last one to AllReduce() were overlapped with GPU computation, and parallel efficiency was very high. We also analyzed scaling the training job to 128 GPUs (16 nodes) connected via a TCP network. For this multi-node case, we found that it was advantageous to specify a ~200 MB bucket when initializing PyTorch DDP, because that reduces the impact of latency in the network. Using the Pytorch profiler, we observed that most of the NCCL communication was overlapped with computation, and the overall parallel efficiency was ~77% relative to a single-node. Higher network bandwidth would result in higher parallel efficiency. With the benefit of overlapping communication and computation on the GPU, good parallel efficiency can be achieved even though TCP protocol provided relatively low network bandwidth, ~6 GB/sec.

Fully Sharded Data Parallel (FSDP). FSDP shards both the model parameters and optimizer states across GPUs. Using FSDP reduces memory consumption per GPU, thus enabling the training of large AI models, but it also increases the communication overhead. The sharding unit is normally one model layer. The parameters for one model layer are gathered and reconstructed on each GPU, and a reduce-scatter operation is used for the gradients of each layer. By careful scheduling of the communication calls, it is often possible to achieve good overlap between communication and computation on the GPU. We investigated training performance using FSDP for the T5 11B model with TCP, RoCE, and GDR communication protocols and varying batch sizes. Measurements are illustrated in Figure . For small batch sizes, the time required for computation is small, and a very high network bandwidth is required in order to achieve good parallel efficiency. At batch size of one, only the GDR method provided good training throughput. As the batch size increases, less network bandwidth is needed because more of the communication can be overlapped with computation on the GPUs. For batch size 32, we observed that TCP could achieve ~92% of the GDR training performance. The choice of batch size depends on the specific requirements of the training job, the available hardware, and trade-offs between training speed, memory usage, and other considerations. However, as illustrated here, in favorable cases it is possible to achieve high training performance with the relatively cheaper and readily available TCP networks in data centers.

3 Experiences

Virtualization Overhead. It is well-known that virtualization (if not appropriately tuned) can adversely impact application performance. However, instead of virtualizing the GPU, we do a full passthrough of the GPU physical PCIe function to the guest OS leveraging the Linux kernel's VFIO subsystem, and from a compute perspective, the LLM training jobs are primarily GPU-bound. So, our first task was to understand and reduce any significant overhead caused by virtualization for GPU-bound jobs. As seen in Figure , for the single-node GPU-bound performance evaluation, we focused on CPU-to-GPU data transfers (d2h and h2d [ ]), intra-GPU memory copies (d2d), GPU kernel execution (matmul, mergesort, eigen), and LLM training (bertlarge, nmt12, megatron). As shown in the figure, the final relative performance achieved inside VMs (after enabling all optimizations) was within a ~5% overhead compared to executing on a bare-metal node. The enablers in achieving the low overhead (as compared to the out-of-the-box virtualization performance) were the backing of the virtual machine memory by huge pages enabled on the hypervisor, ensuring NUMA-GPU mapping in the VM domain XML, and making sure that persistence mode is enabled on the NVIDIA GPUs inside VMs [ ]. The next target was network performance. As the VM network interfaces are SRIOV-based (instead of virtio), the guest OS bypasses the hypervisor networking stack when communicating with the NIC, which enabled us to achieve near-line-rate performance (i.e., within ~2% of hardware's theoretical limit) from within the VMs using iperf [ ] as

we applied TCP optimization inside the guest OS recommended for HPC workloads when operating in public cloud VMs [ ]. However, getting near-line-rate performance for RoCE was non-trivial. It required interaction with the NIC vendor, which resulted in changes in the system BIOS, NIC firmware, network SRIOV-VF's PCIe settings from the hypervisor, and the guest OS. We used linux-rdma/perftest utility to benchmark RoCE performance [ ].

Using micro-benchmarks for TCP and RoCE to achieve near-line-rate performance ensured the infrastructure's readiness. However, multi-node LLM training workloads' performance heavily relies on periodic collective communication calls. NVIDIA Collective Communication Library (NCCL) implements commonly used collective operations in LLM training workloads optimized for NVIDIA GPUs and NICs. We observed significantly lower performance while evaluating the collective call performance inside the VMs. Besides, getting bare-metal-like performance for the collective operations over GDR-based communication also required additional changes on the network SRIOV-VF to enable ATS from the hypervisor and explicit loading of NVIDIA's peermem kernel module after every VM reboot. Figure shows the AllReduce collective throughput with different array sizes for Bare-Metal (BM), Unoptimized VM setup (VM-U), and an Optimized VM setup (VM-O) achieved when using TCP, RoCE, and GDR-based communication – showing that near-BM-like collective operation performance can be achieved with VMs. As can be seen in the figure, the performance from VM-U to VM-O improves significantly for GDR-based communication as we increase the number of network interfaces, as we augment NCCL with the system topology information, and it leverages P2P DMA between the GPUs and the NICs connected on the same PCIe Switch. However, in the case of TCP and RoCE, the improvements were limited by the PCIe Gen3 link between the CPU and the PCIe switches. The impact of low overheads in compute and communication performance was seen in multi-node training performance once we optimized collective communication performance. As seen in Figure , the LLM training performance with VMs was within ~5% of that observed on a bare-metal system as we varied the communication protocols and number of network interfaces.

Virtualized PCIe Topology and NCCL. PCIe topology refers to PCIe devices' physical arrangement and connectivity within a system and plays a crucial role in determining the communication efficiency between PCIe devices. For example, in the case of GDR, it plays a vital role in deciding the path between the GPU and devices. A well-designed PCIe topology can support larger-scale GPU clusters by minimizing contention and maximizing available bandwidth, as it directly influences the efficiency, latency, and scalability of data transfers between GPUs and a RoCE-capable NIC. In Vela, the GPUs and NICs are connected to the same PCIe switch, and the NICs are RoCE enabled, i.e., GDR-based communication is enabled.

In our experience with Vela, we encountered three critical issues when using NCCL and GDR inside the VMs. First, to leverage GDR for a collective call like AllReduce, Nvidia's Collective Communication Library (NCCL) attempts to determine the PCIe topology at runtime based on which to construct the communication graph that is used to establish optimal communication paths. NCCL was unable to determine the correct topology from inside the VMs. When we attempted levering GDR, we noticed that the traffic was going over the PCIe root complex – which we determined by analyzing the PCIe traffic from the hypervisor. In this scenario, the performance was similar to RoCE performance instead of GDR, as shown in Figure (c) (see VM (U) vs VM(O)). Second, NCCL struggled with GDR communication with Dual Port NICs inside the VM. Luckily, NCCL provides hooks to pass an XML format topology file using which we achieved the expected performance for GDR. Third, as NCCL is continuously being developed and optimized, we observed performance regression across different versions of NCCL in the VM environment. The regression issue becomes critical in production environments as Pytorch, by default, comes with a statically linked NCCL version. Because of this, our users have experienced performance regression when using pre-built pytorch container images for their workloads. We've communicated these issues to NVIDIA [ ].

Potential RAS Issue. KVM+QEMU-based virtualization platforms support emulating different chipset models used in modern computer systems, of which the most common emulated chipsets are *i440fx* and *q35*. The *i440FX* chipset is an older, legacy chipset commonly used for virtualization in KVM environments. It emulates hardware features typical of older systems, such as legacy PCI and ISA buses. On the other hand, the *q35* chipset is a more modern and versatile chipset model introduced to support advanced virtualization features and emulate newer hardware. It supports advanced hardware features and configurations such as PCI Express Switches and Slots, Device Hotplug, USB 3.0, SATA III, Secure Boot, Confidential Computing, etc. Thus, *q35* chipset emulation may appear as the obvious choice for better device compatibility and performance in VMs running modern OSes and applications that rely on these advanced features.

In Linux-based hypervisors, it is recommended to passthrough the PCIe device's physical function using the Virtual Function I/O to get the best performance for PCIe devices. As *q35* chipset emulation natively supports PCI Express standard, with PCIe device passthrough, the PCIe device's extended configuration/register space is also available inside the VM. As we explored Vela's architecture, we learned that we could crash the hypervisor inside the VM by fuzzing the GPU's extended configuration/register space – which ca potentially cause a Reliability, Availability, and Serviceability (RAS) issue in cloud deployments. As we further explored,

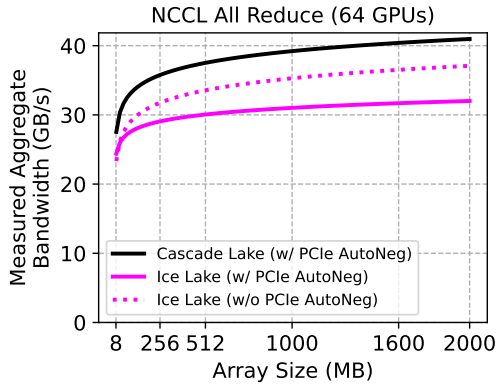
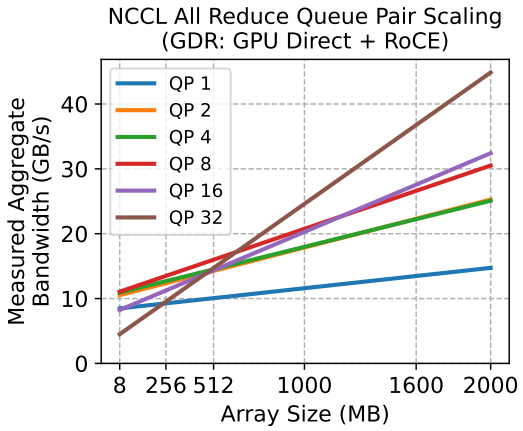


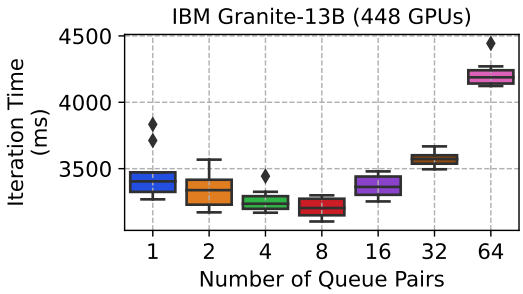
Figure 10. CPU Matters: Cascade Lake vs Ice Lake

we confirmed the extended configuration/register space as an attack surface using other PCIe devices (i.e., NVME drives and NICs). We also confirmed that PCIe devices are presented as legacy PCI devices in passthrough (i.e., the extended configuration/register space is not visible) with *i440fx* and thus are not prone to device fuzzing-based attacks. We are actively working with the Linux community and device vendors towards developing mitigation strategies, and also working on a scientific publication around the issue.

CPU and Vendor Matters. When using GPU Direct, the CPU's involvement is minimized, as it relinquishes control after initializing and coordinating DMA (Direct Memory Access) between the GPU and the target device. As we scaled Vela's deployment, we explored upgrading the CPU from Intel Cascade Lake to Intel Ice Lake generation. In our first attempt, GPU Direct did not work as we observed connection errors when benchmarking the system using the *linux-rdma/perftest* utility. After closely debugging the issue with NVIDIA, we learned that the PCIe Read Completion Block (RCB) boundary, which the CPU had negotiated with the devices at boot time, was not adhered to when IOMMU was enabled. To resolve this issue, we had to obtain an updated BIOS from the server vendor (which included microcode patches from Intel) to ensure GPU Direct works on VMs created on the Intel Ice Lake-based A100 systems. Furthermore, while working on Intel Ice Lake-based systems from a different vendor, we observed ~25% lower performance for NCCL All Reduce operations with GDR than on Intel Cascade Lake-based systems (see Figure). Upon further investigation we observed that as we turned off auto-negotiation on the PCIe links and configured them to their maximum rates, the collective communication performance with GDR improved between ~5%-~25%. As we worked on resolving these issues, we learned that it is common for server vendors not to validate/certify their AI line of servers for virtualization. Even if they validate with virtualization, they primarily focus on VMware or Hyperscaler-specific hypervisors.



(a) NCCL GDR QP



(b) IBM Granite-13B GDR

Figure 11. Queue Pair Exploration

Queue Pair Selection. In Remote Direct Memory Access (RDMA) based communication, queue pairs facilitate efficient data transfers between endpoints. Using queue pairs, RDMA enables direct data transfers between application buffers without involving the CPU, reducing latency and offloading data movement tasks to the network interface controller (NIC). Each RDMA queue pair consists of a send and a receive queue, allowing for bidirectional communication. The send queue holds outbound data messages to be sent to the remote endpoint, while the receive queue holds space for incoming data messages from the remote endpoint. Increasing the number of queue pairs may improve communication latency and throughput through increased concurrency, reduced contention, and better load balancing.

Figure (a) shows AllReduce collective performance for varying array sizes with 256xNvidia-A100 GPUs using GDR on Vela. We observe that using more queue pairs resulted in lower performance for small array sizes but higher performance for large array sizes. However, as we explored increasing the number of queue pairs with real training workloads, we observed that the training performance plateaued around the eight queue pairs and started observing lower performance after that (see Figure (b)) – indicating that the collective calls made by real workloads are towards the

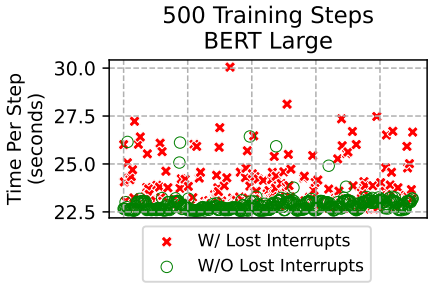


Figure 12. Performance drop due to race condition in kernel.

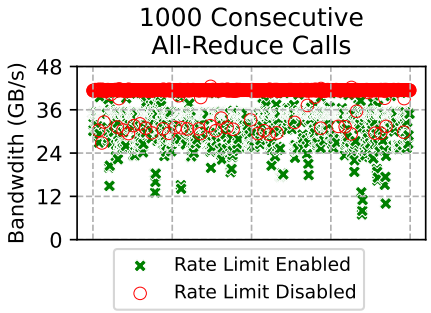


Figure 13. Side Effect from Rate Limiting on VF.

lower end of the array size spectrum shown in Figure (a). As each message is split into N parts and sent on each queue pair, configuring a large number of queue pairs can cause a latency increase (and bandwidth reduction). Especially for small messages, AllReduce collective operation for smaller array sizes gets adversely affected as we increase the number of queue pairs – thus, the training performance gets adversely impacted for the 13B parameter job as we go beyond eight queue pairs. This shows that the decision to increase the number of queue pairs should be carefully considered based on the LLM training workloads.

Linux-kernel as Hypervisor. *(a) Lost Posted Interrupts*: Production environments typically use older but stable software stacks instead of their state-of-the-art counterparts – and this was the case for Vela. In its first release, Vela was integrated into an existing production cloud stack that supported only TCP communication over SR-IOV interfaces for the network traffic between GPU nodes. When we started running training workloads on the system, we observed intermittent workload performance drops – as seen in Figure - some training steps experienced unexpected delays of up to several seconds. We found that this behavior was sensitive to the number of TCP combined channels configured for the multi-queue NICs. In particular, the configuration with the highest maximum bandwidth, channels = 11, caused the largest number of unexpected delays, while limiting the number of channels to 1 minimized the effect, but also lowered the achievable bandwidth. After an in-depth root-cause analysis, we found that the observed behavior was due to a bug in the KVM Linux kernel module on the production hypervisor. Posted Interrupts (PIs) from the SRIOV-VF passthrough devices were getting lost or overwritten due to a race condition [ ] in the kernel. This bug had already been fixed in newer Linux kernel releases. We patched our production hypervisor kernel with a fix back-ported from a newer kernel, and the workload performance improved significantly, as seen in Figure .

(b) ATS Reset: Enabling GDR inside VMs requires configuring the NIC's firmware-level settings. One of these critical settings is Address Translation Services (ATS) – which allows caching of address translations on the PCIe device to prevent going over the PCIe root complex for every address translation as forced by the Access Control Services (ACS). However, as we created SRIOV-VF on the physical NICs, the SRIOV-VFs did not inherit the ATS setting from their parent device and required resetting it from the hypervisor. Furthermore, as we *passthrough* the SRIOV-VF's to the VM (similar to other devices using *vfio*) the SRIOV-VF's gets reset when IOMMU on the hypervisor kernel is set to passthrough mode (i.e. *iommu=pt*) and the MOFED driver inside the VM was unable to detect the ATS setting during the boot time – thus not configuring for GDR inside the VM. As we closely worked with Nvidia to investigate the root cause, they acknowledged that the observed behavior was due to a bug in the Linux kernel [ ]. The Smallest Translation Unit (STU) in PCI Address Translation Services (ATS) is a shared global field set in the Physical Function (PF) but used by all Virtual Functions (VFs). Currently, the STU is only programmed when the IOMMU driver enables ATS, which causes issues if the PF is configured to use IDENTITY translation (e.g., when the device and iommu are in passthrough mode) and the VF would like to use VFIO with a PAGING domain and have ATS turned on. This results in ATS enablement failures for VFs because the PF never loaded the paging domain and thus never programmed the STU. To prevent resetting the ATS on every VM reboot, one can set IOMMU on the hypervisor kernel to no-passthrough mode (i.e. *iommu=nopt*). However, setting the IOMMU in no-passthrough mode on the hypervisor forces all the DMA and Interrupt Remappings to be performed on the hypervisor instead of the VM – in our testing, we did not observe any significant overhead either in GPU operation or GDR performance.

Rate Limiter. It is common to share a physical network interface card (NIC) in a cloud deployment across multiple SRIOV-VFs and apply constraints on them to enable isolation between different entities (i.e., tenants and providers) for reasons related to QoS, cost control, security concerns, etc. In Vela, we share a NIC between hypervisor control-plane traffic and the VM's high-performance RDMA traffic. As we applied rate limiting on the SRIOV-VF used by the hypervisor, the RDMA performance from inside the VMs started experiencing intensive performance drops. Figure

shows the measured bandwidth of 1000 consecutive NCCL All Reduce operations performed from within the VM. As can be seen in the figure, upon enabling the rate limiter on the SRIOV-VFs used by the hypervisor for control plane traffic, we observed up to ~90% performance drop in NCCL All Reduce performance. Upon further investigation, we found that the root cause of the issue was the recently upgraded NIC firmware deployed in the production environment, in which the NIC vendor had introduced new congestion control functionalities. If the rate limiter was disabled on the service VF with the deployed firmware version, the performance improved significantly. The NIC vendor acknowledged the issue and fixed it in one of the subsequent releases of the NIC firmware.

Host Crash with GDR. As we moved from a test environment and formally enabled GDR in our development environment, the hypervisor crashed as we executed *linux-rdma/perftest* to evaluate for GDR. We were puzzled at the time as the crash behavior was not reproducible in the test environment and only observed in the development environment. Upon further investigation, we learned the following: When operating in a multi-tenant cloud environment, it is common for cloud providers to reset device states on the system before provisioning VMs for another tenant – especially critical when provisioning VMs with passthrough devices. Cloud providers often depend on device vendors to provide tools to reset the devices correctly. Thus, in our case, our development team used tools from NVIDIA [ ] that reset the GPUs using PCIe secondary-bus-reset (i.e., *–reset-with-sbr*), which caused a reset of register states (required for GDR) on the PCIe switch without informing the hypervisor, and thus the hypervisor did not restore the correct register states after the reset – resulting in a crash with GDR. To resolve the host-crash issue, NVIDIA suggested resetting the device from Linux-sysfs (i.e., *–reset-with-os*).

Storage vs Training Time: In the first deployment phase, Vela users had the option of using either COS directly or an NFS file system. Compared to NFS performance, the GPFS solution deployed in the second phase achieved ~40x read bandwidth speedup (~1GB/s for NFS vs ~40GB/s GPFS), which directly helps with input data read operations. Also compared to COS bandwidth, the GPFS delivered ~3X write bandwidth speedup (~5GB/s for COS vs ~15GB/s for GPFS), accelerating the checkpoint and other data write operations. These speedups also helped with real-world training jobs. For example, in the case of training 8 billion and 13 billion parameters decoder-only LLMs, we observed the following: (a) Due to the random I/O access patterns, concurrent accesses from multiple reads, and limited data reuse, the LLM training iteration time with NFS takes many steps to reach a steady state. Because of the high bandwidth and low latency performance of GPFS and its ability to handle concurrent accesses, the iteration time reaches a steady state almost instantaneously; (b) During steady state training, multiple clients access the file system simultaneously. Since NFS has limited concurrency support, the step time tends to vary by almost ~50%. In the case of GPFS, the observed step times variations were less than 10%; (c) The higher performance of GPFS translated to ~10% faster LLM training time.

4 Lessons Learned

Below we summarize the lessons learned, insights, and potential research opportunities w.r.t. functionality, performance, and security aspects in virtualized AI training systems based on our experience from building and operating Vela.

a. Power Over-Commitment. As AI systems scale in complexity and capability, power consumption will emerge as a critical determinant of their performance and efficiency. Our research highlights that achieving high system density—essential for maximizing computational throughput—does not have to come at the expense of system resilience. We can effectively manage the power supplies by employing throttling mechanisms to maintain system reliability and fault tolerance (which may not be considered/feasible for traditional HPC deployments). This balance between density and resilience, driven by efficient power management, will be crucial for the future of AI infrastructure. Vela's production deployment has "six servers" per rack compared to the industry norm of "two or three servers" for 20KW racks (see Section .).

b. Passthrough Device Fuzzing. Q35-based chipset emulation for PCIe device passthrough in virtual machines allows guest operating systems direct access to physical hardware, improving performance but exposing a critical security vulnerability. Our research shows that this direct access enables privileged users to write to the extended register space of PCIe devices (see Section) – malicious actors can potentially victimize cloud providers and co-located tenants by fuzzing passthrough devices (e.g., NVMe) and potentially cause RAS issue. Although older i440fx-based emulation could reduce this threat, it would also limit modern features necessary for cloud-native deployments, highlighting a trade-off between security and functionality and the need to vet open-source virtualization stack for vulnerabilities related to state-of-the-art functionalities. We provided the code to fuzz passthrough device to a university's research group; they could successfully replicate the behavior in an NSF cloud testbed. We are actively working with the Linux community and device vendors towards developing mitigation strategies, and also working on a scientific publication around the issue.

c. PCIe Topology Emulation. The growing complexity of compute infrastructure required for AI workloads creates significant challenges in delivering optimal performance. Runtimes like NCCL that are fundamental to AI workloads are increasingly becoming tightly coupled and highly optimized for specific hardware infrastructures to achieve maximum efficiency. Our research demonstrates that virtualizing

these high-performance AI compute servers while still delivering performance at scale is feasible (see Figure). However, it presents several challenges and caveats. Virtualization introduces overhead and potential inefficiencies that can hinder the performance gains that these optimized runtimes are designed to deliver. This complexity also creates an opportunity to rethink and redesign the open-source virtualization layers, making them more suited to the specialized demands of modern AI workloads. By adapting the virtualization stack, we can better accommodate the intricate requirements of AI infrastructure, potentially bridging the gap between the flexibility of virtualized environments and the performance needs of cutting-edge AI applications.

d. Training Over TCP Network. High-performance, low-latency networks like RoCE (RDMA over Converged Ethernet) and InfiniBand (IB) are typically essential for maximizing the efficiency of most AI and high-performance computing (HPC) workloads. These networks reduce communication bottlenecks, enabling faster training times and more efficient distributed processing. However, our research demonstrates that training AI models over a TCP-based commodity-Ethernet data center network is also feasible. While TCP lacks specialized networks' high throughput and low latency, our findings suggest that it can still support practical model training, particularly for specific workloads. This insight opens up new possibilities, especially for environments with limited or cost-prohibitive access to high-performance networking (see Section). Moreover, TCP networks may be particularly well-suited for fine-tuning AI models, where the demands are often lower than in large-scale training. By extending the applicability of AI training to more accessible networking infrastructures, this approach broadens the potential for deploying and refining AI models across a broader range of environments without sacrificing performance for specific use cases. In addition, we present details from Vela's GPU Direct RoCE implementation, an analysis of the performance benefits for collective communication operations and training workloads, and its scaling efficiency.

e. Hardware Vendor Challenges. In our experience, hardware vendors developing AI infrastructure tend to focus primarily on validating their systems for bare-metal deployments or proprietary virtualization solutions. While they cater to the specific needs of large-scale providers, they also need to pay more attention to validating hardware functionality and performance with open-source virtualization platforms. This gap in validation creates a recurring challenge, particularly with each new hardware generation, where even minor hardware modifications (e.g., different CPU versions) can introduce compatibility or performance issues when deployed in open-source environments (see Section). This oversight becomes a significant pain point for users relying on open-source virtualization software, as they face difficulties ensuring that AI infrastructure components perform optimally and reliably. The lack of vendor validation results in additional overhead for users, who must troubleshoot, optimize, and verify their systems independently, diverting time and resources from their primary workloads. Through this paper, we aim to bring greater attention to this issue within the community, advocating for broader validation practices by hardware vendors.

f. Application Performance Tuning. In AI training over RoCE networks, selecting the optimal number of queue pairs (QPs) is crucial for achieving efficient communication performance. Our experience shows that while increasing QPs can enhance performance for large data transfers by enabling greater parallelism, it can also have an significantly adverse impact on training speed and overall system efficiency, highlighting the need for careful QP tuning based on specific workload requirements (see Section). Currently, tuning the number of QPs is manual and nuanced, as optimal settings depend heavily on the underlying network architecture and workload size. This challenge becomes even more pronounced in RoCE environments, where proper entropy balancing and workload distribution are vital to maximizing performance. The manual nature of QP tuning presents an opportunity for the research community to develop automated tuning mechanisms. By designing dynamic auto-tuning systems that adjust parameters like queue pairs in real time based on workload characteristics and network conditions, we can improve the performance, scalability, and efficiency of AI and HPC workloads. Automating these optimizations would reduce the need for labor-intensive manual adjustments and make high-performance AI infrastructure more accessible across different deployment environments.

g. Vendor Agnostic SDN. We demonstrate the viability of a vendor-agnostic, flow-based software-defined networking (SDN) solution deployed directly on compute nodes, designed to adapt to unpredictable traffic patterns in multi-tenant environments. This solution enables secure, high-performance communication between virtual machines while maximizing available bandwidth. By leveraging an endpoint-centric architecture, where network processing occurs on the compute nodes rather than relying on a centralized middleware layer, the design eliminates single points of failure, enhancing both scalability and resilience. This approach ensures isolation for tenant traffic and optimizes network performance through hardware offloading, enabling seamless adaptation to dynamic traffic demands without compromising system efficiency or scalability (see Section).

5 Related Work

Recently, several organizations have published details on the design, implementation, and operational experience of their AI training systems [, , , , , , – , , ,]; however, they differ from Vela in several key compute and

network aspects. For example, companies like Meta [ ], Azure [ ], Amazon [ ], and Google [,] that deploy bare-metal or virtualized systems for AI training employ customized hardware designs and components purpose-built for their specific deployments (e.g., server chassis, topology, network adapters, ASICs) and do not attempt to over-commit power at the rack-level. Moreover, existing public [,] and private cloud [ ] solutions that enable high-performance GPU-VM instances for AI training either employ closed-source hypervisors or do not publish any implementation details if they use open-source technologies. Vela, on the other hand, is built using open-source Linux-KVM and QEMU technologies, and we provide comprehensive information that enables users to set up high-performance GPU VMs for LLM training workloads.

Furthermore, while software-defined networking (SDN) and network offload hardware like AWS Nitro [ ] and Azure Boost [ ] are well-established (though implementation details are not publicly accessible and are dependent on the underlying hardware), this paper details a vendor-agnostic SDN solution with network offloading to enable GPUs to perform GPU Direct RDMA over Converged Ethernet (RoCE) with generally available vendor-agnostic network cards in a virtualized environment. Unlike Azure AI cluster [ ], Meta Supercluster [ ], or Nvidia DGX Pod [ ] that employ Infiniband-based RDMA networks, Vela employs a RoCE network. Other recent studies from Alibaba [ ] and Meta [ ] also explore the use of RoCE-based networks for AI training workloads using rail-optimized network designs which sacrifice high availability for performance. One key aspect of Vela's RoCE-based network is that we use ECMP and achieve both high performance and maintain high availability.

Additionally, most of the recent studies focus on specific components of the systems and do not provide a holistic view of the requirements at different system layers. For example, [ ] discusses queue pair scaling for their RoCE network; however, it does not discuss how it impacts actual training workloads. In our experience, managing these modern GPU-based systems introduces unique challenges, such as issues with PCI-E links, firmware, CPU vendor compatibility, and complexities within the software stack—like the NCCL communication library and interrupt processing. These systems demand specialized expertise to operate effectively. This paper aims to make such knowledge more accessible.

6 Conclusion

In this paper, we present a comprehensive approach for building a cloud-native AI system utilizing readily available hardware and open-source Linux KVM-based virtualization stack, along with a virtualized RDMA over Converged Ethernet (ROCE) network to facilitate the training of AI models. Our method has demonstrated exceptional training performance, scalability, multi-tenancy, and adaptable deployment across IBM Cloud. We have also shared valuable insights on how to minimize virtualization overhead and optimize system and application parameters for enhanced performance. By sharing our experiences, we aim to encourage the development of future AI systems that leverage readily available software and hardware technologies, thereby making AI infrastructure more accessible to a broader community.

7 Acknowledgments

We would like to thank our partners at NVIDIA and our server vendor partners for their constant support during the research, development, and deployment phases of Vela. We also thank and acknowledge the IBM Cloud and IBM Research Emerging Technology Engineering teams for their efforts and support. We also thank and acknowledge our shepherd Jacob Nelson, and the anonymous reviewers of the paper for their valuable feedback towards improving the paper.

A Artifact Appendix

A.1 Abstract

This appendix lists the open-source efforts by IBM to help the community in building and benchmarking high-performance virtualized system for AI training workloads using off-the-shelf hardware and open-source virtualization stack. All the artifacts are publicly available at :

- Details for provisioning GPU Direct RoCE capable VMs on a Linux-KVM-QEMU-based based hypervisor optimized of LLM Training workloads (see "vmsetup" directory).
- Pytorch-based benchmarks for the commonly used collective communication operations employed in LLM training workloads. The key communication collective benchmarks are allreduce-loop.py, allgather-loop.py, and reduce-scatter-loop.py (see "pycommbench" directory).
- Code to replicate the potential RAS issue due to passthrough device fuzzing (see "ras" directory).

A.2 Artifact check-list

- **Hardware:** At-least two Intel Ice Lake or Cascade Lake servers with NVIDIA A100 80GB HGX Platform, ConnectX-6 Dx Network Cards, and Samsung NVMe. The servers can be connected directly or via a RDMA over Converged Ethernet (RoCE) capable network switch.
- **Metrics:** For the collective communication benchmarks, the metric is the measured bandwidth in GB/s.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT License.
- **Workflow framework used?:** Python, Pytorch, MPI (details to run the benchmarks available in the git repository).

References

[1] Zane Adam. 2024. The convergence of HPC and AI: Driving innovation at speed.

[2] AMD. 2024. ROCmRDMA: Remote Device Programming.

[3] Rajesh Anantharaman. 2024. Google Cloud demonstrates the world's largest distributed training job for large language models across 50000+ TPU v5e chips.

[4] AWS. 2024. AWS Nitro System.

[5] Azure. 2024. Azure: NDv2 sizes series.

[6] Ian Buck. 2020. NVIDIA A100 Launches on AWS, Marking Dawn of Next Decade in Accelerated Cloud Computing.

[7] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. 2003. The LINPACK benchmark: past, present and future. Concurrency and Computation: practice and experience 15, 9 (2003), 803–820.

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024).

[9] Daniel Firestone. 2017. {VFP}: A virtual switch platform for host {SDN} in the public cloud. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). 315–328.

[10] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. 2024. Rdma over ethernet for distributed training at meta scale. In Proceedings of the ACM SIGCOMM 2024 Conference. 57–70.

[11] Talia Gershon, Seetharami Seelam, Brian Belgodere, Milton Bonilla, Lan Hoang, Danny Barnett, I Chung, Apoorve Mohan, Ming-Hung Chen, Lixiang Luo, et al. 2024. The infrastructure powering IBM's Gen AI model development. arXiv preprint arXiv:2407.05467 (2024).

[12] Google. 2023. Announcing A3 supercomputers with NVIDIA H100 GPUs, purpose-built for AI.

[13] Hasanin Harkous, Michael Jarschel, Mu He, Rastin Priest, and Wolfgang Kellerer. 2019. Towards understanding the performance of P4 programmable hardware. In 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 1–6.

[14] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In Proceedings of the 50th Annual International Symposium on Computer Architecture. 1–14.

[15] KVM. 2019. KVM: x86: Sync the pending Posted-Interrupts.

[16] Nesreen K Ahmed Le Chen, Akash Dutta, et al. 2024. The Landscape and Challenges of HPC Research and LLMs. (2024).

[17] John Lee. 2023. Microsoft Azure Eagle is a Paradigm Shifting Cloud Supercomputer.

[18] Meta. 2022. Introducing the AI Research SuperCluster — Meta's cutting-edge AI supercomputer for AI research.

[19] Meta. 2022. Open hardware for AI infrastructure.

[20] Meta. 2024. Building Meta's GenAI Infrastructure.

[21] Microsoft. 2022. Hypervisor security on the Azure fleet.

[22] Microsoft. 2024. Microsoft Azure Boost.

[23] Microsoft. 2024. Virtual machines in Azure.

[24] Apoorve Mohan. 2022. NCCL Test Failing with newer NCCL versions for RoCE with Dual HCA inside VMs .

[25] Kief Morris. 2020. Infrastructure as code (2nd ed.). O'Reilly Media.

[26] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In Proceedings of the 49th Annual International Symposium on Computer Architecture. 993–1011.

[27] ESnet: Energy Sciences Network. 2023. iPerf3: .

[28] NVIDIA. 2021. CUDA Samples: .

[29] NVIDIA. 2022. NVIDIA Teams With Microsoft to Build Massive Cloud AI Computer.

[30] NVIDIA. 2023. NVIDIA DGX SuperPOD: Next Generation Scalable Infrastructure for AI Leadership.

[31] NVIDIA. 2024. Driver Persistence Mode: .

[32] NVIDIA. 2024. iommu: Allow ATS to work on VFs when the PF uses IDENTITY.

[33] NVIDIA. 2024. NVIDIA GPU Admin Tool: .

[34] Nvidia. 2024. NVIDIA GPUDirect: Enhancing Data Movement and Access for GPUs.

[35] Greg Pauloski. 2021. BERT-Pytorch: .

[36] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. 2024. Alibaba hpn: A data center network for large language model training. In Proceedings of the ACM SIGCOMM 2024 Conference. 691–706.

[37] Linux RDMA. 2022. Linux RDMA Perftest: .

[38] Amazon Web Services. 2024. Recommended GPU Instances.

[39] Hamid Shojanazeri. 2022. Getting Started with Fully Sharded Data Parallel (FSDP) .

[40] SIGPCI. 2024. PCI-SIG Specifications.

[41] A Tekin, A Tuncer Durak, C Piechurski, D Kaliszan, F Aylin Sungur, F Robertsén, and P Gschwandtner. 2021. State-of-the-art and trends for computing and interconnect network solutions for HPC and AI. (2021).

[42] VMWare. 2024. VMware EXSi.

[43] Robert Walkup, Seetharami R Seelam, and Sophia Wen. 2022. Best practices for HPC workloads on Public Cloud platforms: A guide for computational scientists to use public cloud for HPC workloads. In Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering. 29–35.

[44] Yan Yan, Arash Farhadi Beldachi, Reza Nejabati, and Dimitra Simeonidou. 2020. P4-enabled smart nic: Enabling sliceable and service-driven optical data centres. Journal of Lightwave Technology 38, 9 (2020), 2688–2694.