Alibaba Stellar: A New Generation RDMA Network for Cloud AI

Jie Lu*, Jiaqi Gao*, Fei Feng*, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, Dennis Cai
Alibaba Cloud

ABSTRACT

The rapid adoption of Large Language Models (LLMs) in cloud environments has intensified the demand for high-performance AI training and inference, where Remote Direct Memory Access (RDMA) plays a critical role. However, existing RDMA virtualization solutions, such as Single-Root Input/Output Virtualization (SR-IOV), face significant limitations in scalability, performance, and stability. These issues include lengthy container initialization times, hardware resource constraints, and inefficient traffic steering. To address these challenges, we propose Stellar, a new generation RDMA network for cloud AI. Stellar introduces three key innovations: Para-Virtualized Direct Memory Access (PVDMA) for on-demand memory pinning, extended Memory Translation Table (eMTT) for optimized GPU Direct RDMA (GDR) performance, and RDMA Packet Spray for efficient multi-path utilization. Deployed in our large-scale AI clusters, Stellar spins up virtual devices in seconds, reduces container initialization time by 15 times, and improves LLM training speed by up to 14%. Our evaluations demonstrate that Stellar significantly outperforms existing solutions, offering a scalable, stable, and high-performance RDMA network for cloud AI.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Data center networks**; **Network components**;

KEYWORDS

Data center networking, Network support for AI and machine learning applications, Transport and congestion control.

ACM Reference Format:
Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, Dennis Cai. 2025. Alibaba Stellar: A New Generation RDMA Network for Cloud AI. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3718958.3750539

*Authors contributed equally to this paper.

SIGCOMM '25, September 8–11, 2025, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/25/09.
https://doi.org/10.1145/3718958.3750539

1 INTRODUCTION

Recent years have seen significant advancements in Artificial Intelligence (AI), driving rapid growth in demand for cloud-based AI training and inference, particularly for Large Language Models (LLMs). The performance of these models heavily relies on Remote Direct Memory Access (RDMA) [4] to accelerate communication between computational devices like Graphics Processing Units (GPUs). Consequently, the performance of RDMA virtualization in the cloud is on the critical path for effective LLM training and inference.

As one of the largest public cloud providers, Alibaba has been deploying large-scale RDMA networks and serving massive AI applications since 2019. However, we have observed that the current RDMA solution has inherent limitations in providing high-performance, stable, and scalable cloud-based AI services:

First, at the **host-level**, the current RDMA virtualization solution lacks scalability and is not lightweight. The number of Virtual Functions (VFs) an RDMA Network Interface Card (RNIC) can create is static; it can only be toggled between zero and a fixed maximum, making it impossible to dynamically scale the number of VFs to meet the demand for AI/RDMA service expansion. Additionally, we use the RunD secure container [20] to provide stronger security and isolation for our customers. To ensure the correctness of RDMA operations for VFs within a secure container, the container must pin all of its memory in the host memory before initiating any RDMA operation. This memory-pinning operation introduces a prohibitive, minute-level start-up delay.

Second, at the **PCIe-level**, the current RDMA solution suffers from PCIe fabric constraints. AI applications leverage GPU Direct RDMA (GDR) [2] to accelerate communication between GPUs. A VF using GDR requires the device to be registered in the PCIe switch's Look-Up Table (LUT). However, the LUT in PCIe fabrics is severely limited in size, allowing only a small number of VFs to enable GDR. Furthermore, the limited PCIe fabric capability of certain server models forces us to sacrifice non-RDMA traffic performance to achieve GDR performance.

Third, at the **RNIC-level**, the current solution fails to provide stable, high-performance RDMA services. Off-the-shelf solutions configure hardware steering rules to interconnect VFs in secure

containers. However, these hardware steering rules do not support strict isolation between RDMA and non-RDMA traffic. Consequently, the performance of RDMA traffic is often degraded by operations on rules targeting non-RDMA traffic. Moreover, while the AI computation clusters we have built are designed with substantial network bandwidth and a large number of equivalent network paths, RDMA cannot utilize these paths uniformly or optimally.

To eliminate the drawbacks of the current RDMA framework mentioned above, we propose Stellar, a next-generation RDMA solution for cloud-based AI training and inference. Aiming to provide extreme RDMA performance in the cloud, Stellar fuses the design philosophies of HyV [26] and MasQ [14], gets rid of SR-IOV, and introduces the following new designs at the host, PCIe, and NIC levels, respectively:

- **Para-Virtualized Direct Memory Access (PVDMA).** With on-demand memory pinning, PVDMA significantly reduces host memory consumption and greatly mitigates the start-up delay of secure containers.
- **Extended Memory Translation Table (eMTT).** Stellar extends the Memory Translation Table (MTT) in the RDMA NIC (RNIC) to record the device type of a memory address. This allows the RNIC to bypass unnecessary consultations of memory address mappings in the PCIe fabric and consistently provide excellent GDR performance.
- **RDMA Packet Spraying.** Stellar introduces a native multi-path RDMA solution to take full advantage of available equivalent network paths. Stellar can intelligently spray RDMA packets across these paths and robustly handle out-of-order packets.

Without SR-IOV, there is no need to steer RDMA traffic from containers using hardware flow rules, as VFs are absent. Thus, Stellar is inherently free from both the VF scalability problem and the hardware flow isolation problem.

We have implemented and deployed Stellar atop an FPGA-based 400G RNIC in our serverless AI platform for over one year. According to online monitoring statistics, Stellar reduces container initialization time by 15× and increases average RDMA throughput by 1.37%. Furthermore, Stellar decreases the switch queue length by 90% and improves the average training speed by 14%.

The remainder of this paper is organized as follows: Section 2 provides background on networking stack virtualization. Section 3 describes the motivation for designing Stellar and offers insights into our design. Section 4 gives an overview of the Stellar solution, while Sections 5, 6, and 7 illustrate its technical highlights. Section 8 validates the superiority of Stellar through comprehensive experimental evaluation. Finally, Section 9 discusses related topics, and Section 10 concludes the paper.
This work does not raise any ethical issues.

2 BACKGROUND

Virtualization is a critical and fundamental technique for cloud providers to offer customers secure and isolated environments. In this section, we briefly explain the key techniques for building a virtualized, high-performance networking stack.

Secure Containers. RunD [20] is a lightweight secure container runtime that achieves the same level of isolation as traditional virtual machines (VMs). It creates a MicroVM [11] that hosts a Linux kernel atop a hypervisor layer. Programs running in a secure container are isolated from other co-hosted tenants, making it a well-suited runtime for public cloud services.

Memory Mapping Hierarchy. Figure 1(a) illustrates the multi-layered memory address translation from an application in a RunD container, through the hypervisor and host OS, to the physical main memory. An application running within a RunD container uses Guest Virtual Addresses (GVA). These are translated to Guest Physical Addresses (GPA) by the guest OS's page tables (PTs). From the guest's perspective, a GPA appears to be a true "physical" address, although it is still virtualized. The host operating system (Host OS) then interprets GPAs as Host Virtual Addresses (HVA), which are in turn translated into Host Physical Addresses (HPA) by standard page tables within the host OS.

The HVA→HPA mapping is managed and accelerated by the Memory Management Unit (MMU) in the Root Complex (RC). The GPA→HPA mapping involves two levels of indirection, which can slow down applications within the RunD container. Modern CPUs extend the MMU with Extended Page Tables (EPT) to directly map GPA to HPA in hardware. To leverage this feature, the hypervisor registers the container's memory space with the MMU. During runtime, the CPU monitors the container's memory usage and caches the corresponding GPA-to-HPA mappings in the EPT.

PCIe devices can also issue memory accesses to main memory to perform operations such as RDMA. First, the Host OS allocates a memory region in the HVA space and determines its corresponding HPA mapping. The PCIe device driver then creates a Device Address (DA), notifies the device, and programs the Input-Output Memory Management Unit (IOMMU) to map the DA to the HPA. Once setup is complete, the device can access main memory by sending requests that target the DA space; the IOMMU then translates these DAs to HPAs.

PCIe Subsystem. Figure 1(b) illustrates the PCIe subsystem's topology and communication workflow. The CPU relies on PCIe Base Address Registers (BARs) to enable communication with PCIe devices. Each PCIe device has a dedicated Bus-Device-Function (BDF) identifier. Each BAR defines a memory-mapped region in HPA that the CPU can directly access. For example, in Figure 1(b), the GPU's BAR maps a device memory region to addresses 0x00–0x0F in the HPA space. When the CPU issues a memory write request targeting this region (①), the PCIe RC and switch forward the request to the GPU. Conversely, a request targeting a different HPA range, such as ②, is forwarded to main memory.

One PCIe device can communicate with another via PCIe Peer-to-Peer (P2P) technology. In a P2P transaction, a source PCIe device generates a message targeting the address space registered in the target PCIe device's BAR. For example, in request ③, the RDMA-capable NIC (RNIC) sends a request to the GPU by targeting an address in the GPU's registered BAR space (*i.e.*, 0x00–0x0F).

Device-to-main-memory communication follows the same principle. For instance, in ④ of Figure 1(b), if the RNIC needs to read from or write to main memory, it generates a request with an

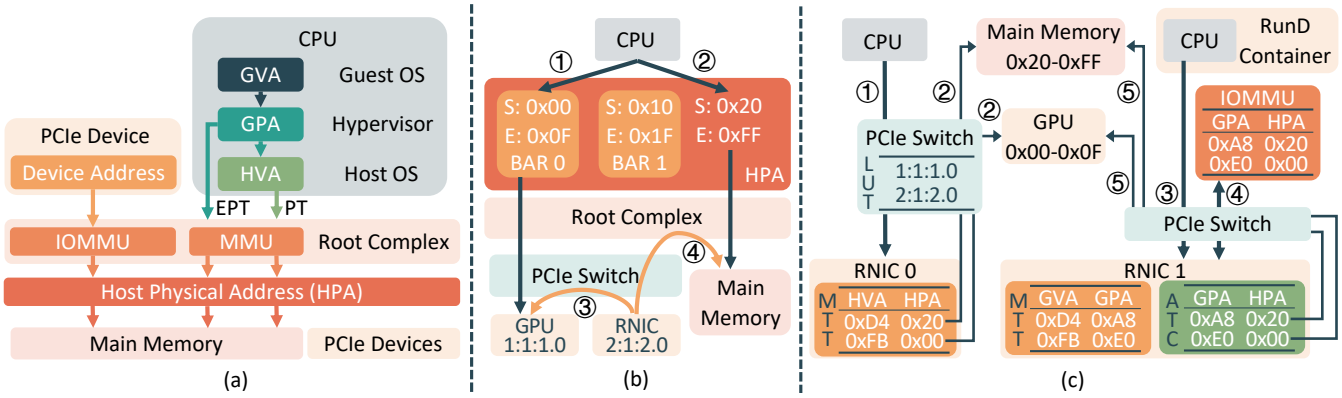


Figure 1: (a) Memory mapping hierarchy. (b) PCIe subsystem and CPU addressing. (c) RDMA and GDR workflow.

address within the HPA range mapped to that memory (e.g., 0x20–0xFF).

RDMA and GPUDirect RDMA (GDR). Figure 1(c) illustrates the RDMA and GDR mechanisms in both physical and RunD virtualized environments. In a physical environment, an application registers a memory region in the HVA space that the NIC can access, which can point to either main memory or a GPU (step ①). The RNIC driver then retrieves the HVA-to-HPA mapping and writes it into an RNIC-internal data structure called the Memory Translation Table (MTT). For example, address 0xD4 may map to 0x20 in main memory, and 0xFB may map to 0x00 on the GPU. Later, when an RDMA or GDR request arrives targeting an HVA, the RNIC uses the MTT to translate the address to the corresponding HPA before the PCIe switch and RC forward the request to its destination.

In a RunD environment, the container's driver is unaware of the GPA-to-HPA mapping. Therefore, it writes the GVA-to-GPA mapping into the MTT (step ③) and relies on the IOMMU in the RC for the subsequent GPA-to-HPA translation. The hypervisor registers this GPA-to-HPA mapping in the IOMMU. When an RDMA or GDR request with a GVA arrives, the RNIC first queries the IOMMU's Address Translation Service (ATS) to resolve the HPA, which is then returned to the RNIC (step ④). The RNIC caches this translation result in its internal Address Translation Cache (ATC) to avoid the performance bottleneck of subsequent IOMMU queries. Finally, by checking the MTT and ATC, the RNIC completes the full GVA→GPA→HPA address translation and issues the request to the target hardware (step ⑤).

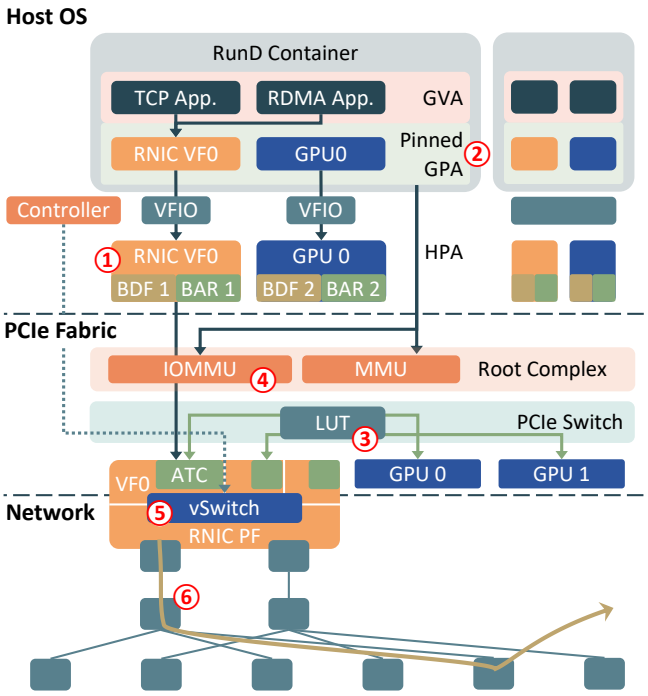


Figure 2: The current network virtualization framework and issues we encountered during operation.

3 CURRENT VIRTUALIZATION SOLUTION

In this section, we introduce our state-of-the-art solution for AI computing clusters in the public cloud, which uses off-the-shelf technologies. We also share several key challenges we have encountered in recent years. Figure 2 illustrates the RDMA virtualization framework.

The host OS runs one or more RunD containers, each equipped with at least one NIC and one GPU. The NIC inside each container (*e.g.*, RNIC VF0) is a Virtual Function (VF) created from the RNIC's Physical Function (PF) using SR-IOV technology. It has a dedicated PCIe BDF and BAR space. The PCIe devices (*e.g.*, RNIC, GPU) are assigned to the container using the Virtual Function I/O (VFIO) driver. VFIO maps the PCIe hardware resources (*e.g.*, BARs) from HPA to the RunD container's GPA space by configuring the IOMMU in the RC. Applications inside the container can then access the device as if running on bare metal, providing optimal RDMA/GDR performance.

On the PCIe fabric, as introduced in Section 2, the IOMMU, ATC, and the PCIe switch's LUT are configured to enable RDMA and GDR for RunD containers.

The RNIC leverages its built-in vSwitch to direct traffic between the RNIC PF and its VFs. This vSwitch is controlled by a Controller

process in the host OS, which is maintained by our container networking team. The Controller maintains a complex VxLAN-based virtual-to-physical network mapping. Since this mapping's requirements exceed the vSwitch's capacity, the Controller tracks the active network connections of each container and dynamically offloads relevant rules to the vSwitch. For each active connection, the RNIC randomly chooses one of its two ports to send traffic to the network. All packets within the same connection have the same header and follow the same path.

3.1 Experiences and Lessons

Over years of operating our large-scale AI cluster with this virtualization framework, we have experienced problems across all dimensions of the architecture. Below, we review six representative problems, with corresponding components marked in Figure 2. Problems ① and ② reveal host-level inflexibility caused by SR-IOV and the VFIO driver. Problems ③ and ④ stem from hardware resource and functionality limitations in the PCIe fabric. Problems ⑤ and ⑥ are rooted in the tight coupling between the RNIC's RDMA and TCP flow steering mechanisms.

① **VF Inflexibility.** The VF solution from our RNIC vendor does not support dynamic reconfiguration. Specifically, we cannot adjust the number of enabled VFs from one non-zero value to another without a full reset. For example, if two VFs are already enabled, we cannot create an additional VF directly. We must first remove all existing VFs and then create three new ones. This constraint forces us to configure the number of VFs on the RNIC only during host startup. Overprovisioning VFs is not feasible, as each VF claims 63 virtual queues of 5000 MTU messages each, consuming 2.4 GB of memory in total. Naively increasing the total number of VFs would create a formidable memory overhead.

② **Pinned GPA Required by VFIO.** We found that the RunD container start-up time is unexpectedly long, primarily due to MMU interactions. Normally, VFIO is performant because it configures the IOMMU to map a PCIe device's BAR from GPA to HPA, allowing applications to operate the device directly. However, in a RunD container, this mapping is not static. When the host OS swaps out HPA memory pages, the GPA-to-HPA mapping changes, causing the RNIC driver inside the RunD container to behave unpredictably and crash. The workaround is to configure the MMU in the RC to pin these memory regions, preventing them from being swapped out. Since AI applications commonly use RDMA and GDR, the RunD hypervisor must pin both the memory region marked by the VF's BAR and all potential memory regions used by RDMA, which effectively means all memory inside the RunD container. Pinning a container with 1.6 TB of memory typically takes 390 seconds, which significantly delays container start-up.

③ **PCIe Switch LUT Capacity Limitation.** In our LLM inference cluster, dense container deployments (over 100 per server) often result in instances that cannot enable GDR. Our investigation revealed that the issue lies with the PCIe switch's limited capacity. Each VF has a unique BDF in the PCIe subsystem. As introduced in Section 2, GDR requires registering the VF's BDF in the PCIe switch. However, on one of our AI cloud server models, each PCIe switch can only accommodate 32 BDFs. The server contains four RNICs, four PCIe switches, and eight GPUs. This means each RNIC can enable at most eight VFs, leading to a maximum of 32 VFs per server, a number far below our deployment density requirements. While in production, GPU servers often support hundreds of virtual instances [5, 6].

④ **Conflicting PCIe Fabric Settings.** A customer reported an abnormal TCP performance issue on a specific server model. Both GDR and VFIO rely on PCIe ATS and IOMMU functionality. On this model, we cannot enable PCIe ATS when the IOMMU is set to `pt` in the host OS kernel. Although we suspect this is related to CPU or OS kernel settings, the root cause remains unidentified. To guarantee GDR performance in RunD containers, we enabled ATS and set the IOMMU to `nopt` in production. However, this setting degraded the host OS's TCP performance because the kernel's TCP stack had to use the RNIC's I/O Virtual Address as the DMA address, creating a performance bottleneck.

⑤ **Interference in RNIC Hardware Flow Steering.** In our current network virtualization framework, TCP and RDMA traffic traverse the same hardware flow steering pipeline in the RNIC's vSwitch. This creates unnecessary interference between the two traffic types, where incorrect TCP processing logic can negatively affect RDMA traffic. We share two issues triggered by this interference.

First, in one cluster, we found that RDMA traffic experienced higher-than-normal latency. Analysis revealed the root cause was the order of entries in the RNIC's vSwitch. TCP traffic entries were placed before RDMA entries, causing RDMA packets to experience a longer hardware lookup time. In this scenario, since all containers share a set of hardware table entries, the RDMA performance of one container can be impacted by the TCP traffic of others.

Second, in another example, we found that two VFs on different RNICs on the same server could not communicate using RDMA. The problem lay in the coordination between the RNIC driver and the Controller. When an RDMA connection is established, the Controller installs a VxLAN encapsulation entry in the vSwitch. The RNIC driver then looks up its routing table to fill the VxLAN header's MAC address field. Because the two VFs belonged to the same server, the driver found a local forwarding rule and set the source/destination MAC addresses to zero. However, since the VFs were on different RNICs, they could only communicate via the ToR switch. The ToR switch, in turn, treated these packets as corrupt and discarded them. Here, the driver's behavior was correct for kernel protocol stacks but incorrect for the RDMA protocol.

⑥ **Single-Path RDMA Transmission.** To support a larger network scale and reduce traffic collisions, we adopted a dual-plane, rail-optimized [27] network topology. In our topology, both planes are connected at the core switch to create an "escape" layer for failure resiliency. In our multi-tenant cluster, we found that the core switch layer created a serious hash imbalance problem when the cluster scheduler deployed an LLM training task across multiple pods. The fundamental reason is that the RNIC does not support multi-pathing; all RDMA packets from the same connection have the same header and traverse the same path. A hash imbalance can easily create network bottlenecks and degrade the performance of AI tasks.

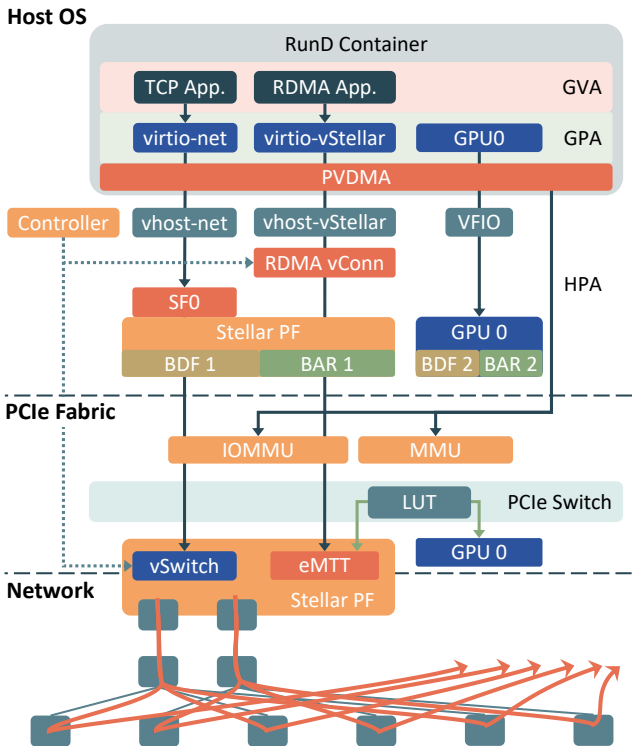


Figure 3: Overview of Stellar design.

3.2 Insights

High-performance transport protocols such as RDMA and GDR are the cornerstone of AI training and inference. The six examples above clearly illustrate the mismatch between current RDMA solutions and the demands of public cloud services. The fundamental issue is that current virtualization solutions depend on the flawless collaboration of numerous software and hardware components. A failure or limitation in any single component can compromise the entire system. This operational experience convinces us that it is necessary and urgent to design a new high-performance RDMA framework tailored for AI applications in the public cloud. In summary, this framework must satisfy the following requirements:

- **Agile Virtualization.** The RNIC must be able to create and destroy hundreds of virtual devices dynamically, and these virtual devices must have a fast boot-up time.
- **Stable Performance.** RDMA and GDR performance must remain stable, regardless of the number of virtualized devices or management events related to non-RDMA traffic.
- **Multi-path Support.** The RNIC must fully leverage the abundant equivalent paths in the network to achieve better and more stable RDMA/GDR performance.

4 STELLAR: RDMA FOR THE AI CLOUD

Since 2022, we have been designing and developing Stellar, the next-generation RDMA virtualization framework for AI workloads in the cloud. In Stellar, RDMA is treated as a first-class citizen. Stellar overcomes the drawbacks of state-of-the-art RDMA virtualization solutions to achieve scalability, stability, and performance simultaneously. Figure 3 provides an overview of the Stellar design.

In a secure container, Stellar provides two Virtual I/O (virtio) [32] devices to handle RDMA and other traffic separately. `virtio-vStellar` takes over all RDMA traffic, while `virtio-net` is responsible for the remaining network traffic, such as TCP, User Datagram Protocol (UDP), and Address Resolution Protocol (ARP). For the remainder of this paper, we use TCP to represent all non-RDMA traffic.

For TCP traffic, Stellar supports off-the-shelf technologies, including `virtio-net`, virtio Data Path Acceleration (vDPA), PCIe Scalable Functions (SFs), and VxLAN tunneling. We chose SFs because they are more lightweight and flexible than VFs, allowing for dynamic creation and deletion and thus solving the VF agility issue. While the virtio/SF/VxLAN solution incurs a performance penalty of approximately 5% compared to the vfio/VF/VxLAN approach, its impact is minimal. This is because TCP traffic in distributed AI/LLM workloads typically serves as control messages, which have a negligible impact on end-to-end job performance.

For RDMA traffic, we implemented a hybrid virtualization solution named vStellar. In vStellar, the RDMA control path uses virtio; control messages (*e.g.*, Queue Pair (QP) creation, query, and modification; Memory Region (MR) registration) first reach the container's virtio driver. The host virtio driver then intercepts these requests to apply security and virtualization-related functions. The vStellar data path is implemented through direct memory mapping, following the approach of HyV [26] and MasQ [14]. Specifically, the secure container can directly access the RNIC's Doorbell register (vDB), and the RNIC can directly read from and write to the MRs registered inside the container. This design maintains RDMA performance without compromising security.

The use of SFs and the vStellar design in Stellar does not require additional BDFs in the PCIe subsystem. Therefore, all virtual devices can support GDR communication, which solves both the VF scalability issue and the LUT capacity limitation. vStellar devices can be created and destroyed in seconds and consume minimal hardware resources. Note that the functions described thus far are similar to those in previous solutions such as HyV and MasQ.

However, three key challenges remain unsolved by these prior approaches. First, the secure container must still pin all its memory to guarantee that applications can communicate correctly with the GPU and NIC, which leads to long container start-up times. Second, an increase in the number of virtual devices reduces the per-device ATC allocation, which in turn increases the ATC cache miss rate and degrades GDR performance. Lastly, the host-side RDMA virtualization in HyV and MasQ cannot address the performance issues caused by single-path RDMA transmission.

To address these challenges, Stellar introduces three novel solutions:

- On the host, we propose GDR-capable Para-virtualized Direct Memory Access (PVDMA) to enable on-demand memory pinning and registration, thereby accelerating container start-up.

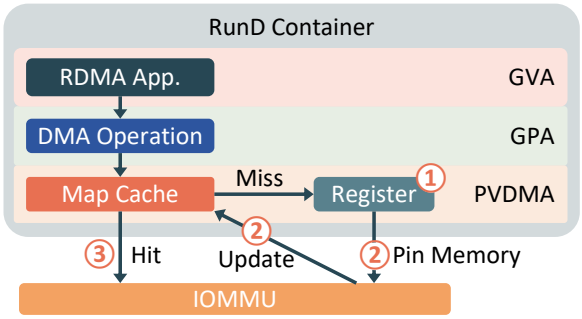


Figure 4: The PVDMA workflow.

- On the RNIC, we extend the MTT to eMTT (extended MTT) to access GPU memory directly. The eMTT bypasses the PCIe RC and avoids the ATC cache miss issue.
- In the network, we spray RDMA packets across 128 paths using an Oblivious Packet Spraying (OPS) algorithm.

Combined, these features allow Stellar to support up to 64k virtual devices, create a new vStellar device in 1.5 seconds (matching the performance of MasQ), accelerate container start-up by 30×, and improve LLM training performance by up to 14%. In the following sections, we explain the design and implementation of these key features in detail.

5 PVDMA: PARAVIRTUALIZED DIRECT MEMORY ACCESS

Paravirtualization [7] is a virtualization technique that allows the guest OS to interact directly with the hypervisor to achieve higher performance and enable custom functionality. We designed a Paravirtualized Direct Memory Access (PVDMA) mechanism to enable on-demand IOMMU address registration and eliminate the upfront GPA pinning overhead.

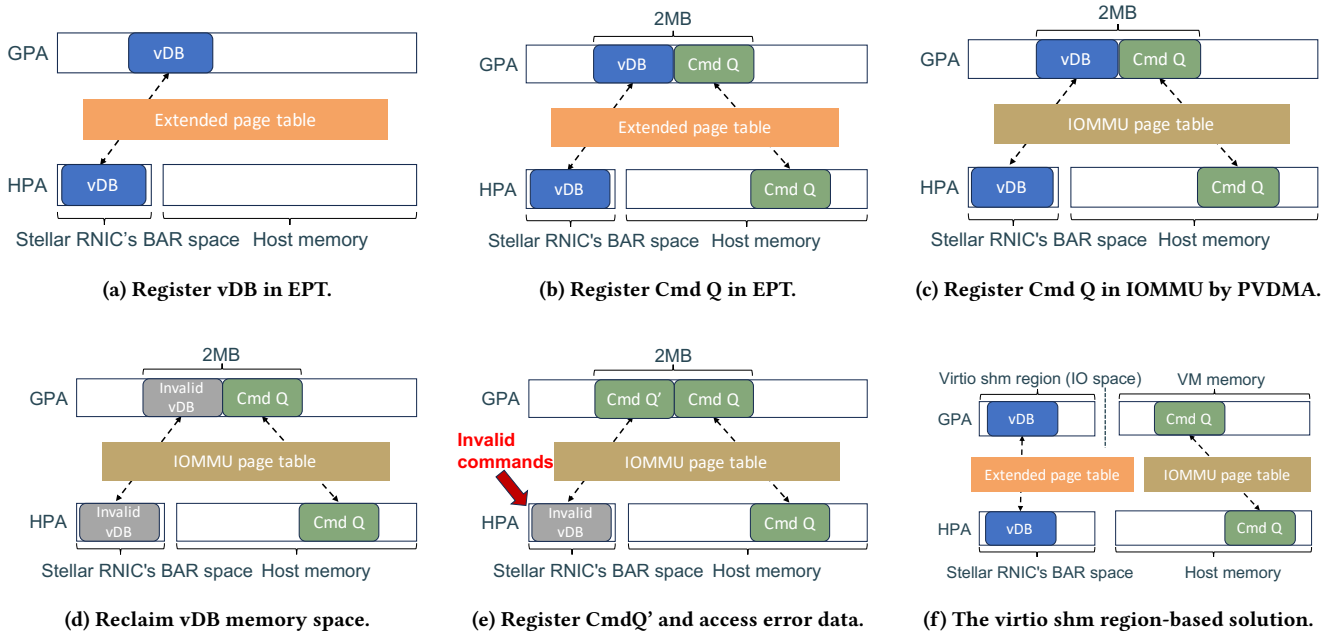
The PVDMA workflow is illustrated in Figure 4. Unlike traditional methods, no memory is pinned during container start-up. When the `virtio-Stellar` driver in a RunD container initiates a DMA operation, PVDMA detects and intercepts the request (stage 1), registers the requested GPA memory region in the hypervisor, and communicates with the IOMMU to pin the corresponding memory in HPA (stage 2). Subsequent DMA operations targeting the same memory region hit the Map Cache and can be executed correctly, as the pinning protects the memory from page swapping (stage 3). Since RDMA applications commonly reuse pinned memory regions, the one-time cost of PVDMA's on-demand pinning does not add noticeable overhead. Furthermore, the Map Cache lookup is lightweight and incurs only negligible latency. This design is inspired by vIOMMU [9] and coIOMMU [31] but offers a more lightweight implementation.

Our dynamic, on-demand PVDMA solution, while avoiding full GPA pinning, introduces the risk of the GPU accessing NIC memory unexpectedly. The fundamental reason is that PVDMA creates a dynamic GPA-to-HPA mapping that can conflict with the MMU's existing mappings for device registers. This issue, illustrated in Figure 5, is triggered by the following sequence of events:

- *Step 1: Registering vDB in the EPT.* When an RDMA program starts, the `virtio-Stellar` driver performs direct memory mapping for the vStellar device's virtual Doorbell (vDB). As shown in Figure 5a, the RNIC's physical Doorbell (DB) register in the HPA BAR space is mapped to the vDB by creating an entry in the MMU's EPT.
- *Step 2: Registering the Command Queue in the EPT.* To allow the CPU to issue commands to the GPU, the GPU driver allocates memory for a command queue (`Cmd Q`). In the case shown in Figure 5b, it accidentally allocates this memory in a GPA region adjacent to the vDB.
- *Step 3: Registering the Command Queue in the IOMMU.* When the GPU is about to read commands from the `Cmd Q` via DMA, PVDMA registers the GPA-to-HPA mapping for this memory region in the IOMMU, as shown in Figure 5c. PVDMA operates with a memory granularity of 2 MiB. Unfortunately, this 2 MiB memory block also covers the vDB.
- *Step 4: Incorrect Retention of vDB Mapping.* After the RDMA program finishes, the EPT mapping for the vDB is released. However, if the GPU application is still running, an inconsistency arises. As shown in Figure 5d, because the memory for `Cmd Q` is still in use by the GPU, PVDMA does not unmap the 2 MiB block from the IOMMU, leaving the stale vDB-to-RNIC-DB mapping intact within the IOMMU page table.
- *Step 5: Erroneous Data Access.* As shown in Figure 5e, the OS may later reuse the original vDB memory space to create a new command queue (`Cmd Q'`) for the GPU. When PVDMA detects that the 2 MiB memory region containing `Cmd Q'` is already registered in the IOMMU, it does not update the page table. At this point, the IOMMU page table incorrectly maps `Cmd Q'` to the physical RNIC DB. Consequently, whenever the GPU attempts a DMA operation on `Cmd Q'`, it inadvertently accesses the RNIC DB, leading to invalid commands and unrecoverable system errors.

The key to solving this problem is avoiding memory address overlap between the MMU-based direct mapping for device registers and the IOMMU-based mapping for PVDMA. The page size for the former is 4 KiB (to reduce hardware resource waste for doorbell registers), while PVDMA's page size is 2 MiB (to balance Map Cache size and IOMMU pinning overhead). Resolving this would require either expanding the doorbell register to 2 MiB or reducing the PVDMA block size to 4 KiB. Neither option is desirable: a larger doorbell register would increase each virtual device's resource consumption, reducing scalability, while a smaller PVDMA block size would increase IOMMU configuration overhead.

Our solution leverages the shared memory (shm) region feature of the virtio framework [32]. This feature provides an I/O space for the virtio device that is distinct from the main physical memory address space. As shown in Figure 5f, we map the vDB into this I/O space. Because this I/O space does not overlap with the physical memory address space used by PVDMA, the conflict is eliminated. However, since this I/O space is not registered in the IOMMU by default, the GPU cannot access it via DMA, which poses a challenge for GPUDirect Async [8]. To support GPUDirect Async, we modified the RunD hypervisor with a mechanism similar to PVDMA



(a) Register vDB in EPT.

(b) Register Cmd Q in EPT.

(c) Register Cmd Q in IOMMU by PVDMA.

(d) Reclaim vDB memory space.

(e) Register CmdQ' and access error data.

(f) The virtio shm region-based solution.

Figure 5: A case of PVDMA causing the GPU to access incorrect data.

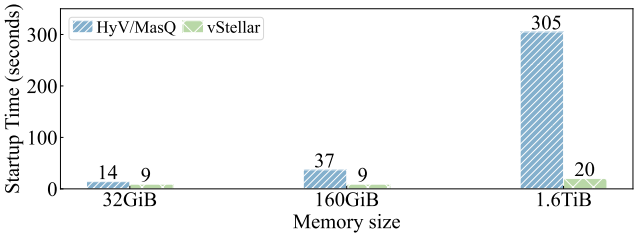


Figure 6: GPU Pod startup time

that explicitly registers the doorbell's I/O memory in the GPU's IOMMU page table when needed.

Figure 6 shows the impact of PVDMA on RunD container start-up time. Without PVDMA, the start-up time increases dramatically with the amount of container memory, as the hypervisor spends most of its time interacting with the IOMMU to pin the GPA. When PVDMA is enabled, the boot time remains below 20 seconds in all cases and is reduced by up to 15×. The slight increase in boot time (11 seconds) between the 160 GB and 1.6 TB configurations is attributable to general hypervisor overhead, not vStellar.

6 GDR VIA eMTT

As mentioned in Section 2, the PCIe ATC is used to cache IOMMU translation results to avoid performance degradation from frequent IOMMU queries. However, the ATC has a limited capacity. According to our experience, an ATC can only cache mappings for tens of thousands of memory pages. When a cache miss occurs, the RNIC must issue additional PCIe requests to the IOMMU to retrieve address translation results, leading to performance fluctuations.

Upon further investigation, we realized that the MTT-ATC two-layer indirection is often unnecessary, as both layers perform memory address mapping. We can directly cache the final translation results in the MTT to bypass the ATC miss problem. The MTT, residing in the RDMA NIC, commonly has orders of magnitude larger capacity than the PCIe ATC. Therefore, we extended the MTT, naming eMTT, to implement this functionality. The eMTT not only records the memory mapping but also stores the memory's owner (*i.e.*, Main Memory or GPU). Stellar handles these two types of DMA operations differently, as illustrated in Figure 7:

For a GDR write operation (GVA=0x20000, key=1), the processing steps are as follows:

- *Step 1:* The packet arrives at the RNIC's RX pipeline. Based on the GVA-to-HPA mapping in the eMTT, the destination HPA is set to 0xA000. Since the target memory type is GPU, Stellar sets the Address Translation (AT) field of the PCIe TLP to `0b10` (translated) [3].
- *Step 2:* When the packet arrives at the PCIe switch, its AT field of `0b10` causes the switch to route the packet directly to the target GPU, bypassing the RC.

For an RDMA write operation targeting host memory (GVA=0x1000, key=0):

- *Step 1:* The packet arrives at the RNIC's RX pipeline. Based on the GVA-to-HPA mapping in the eMTT, the destination HPA is set to 0xB0000. Since the target memory type is host memory, Stellar sets the AT field of the PCIe TLP to `0b00` (untranslated).
- *Step 2:* The PCIe switch inspects the AT field, sees `0b00`, and routes the packet to the RC.
- *Step 3:* The IOMMU in the RC performs the final address translation, and the packet is ultimately routed to host memory.

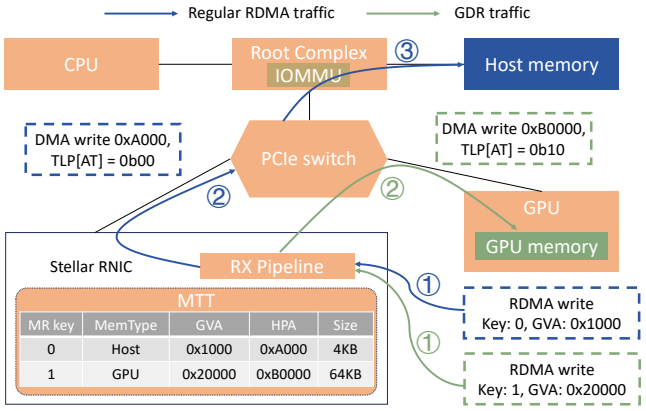


Figure 7: An example of *Stellar* handling GDR traffic.

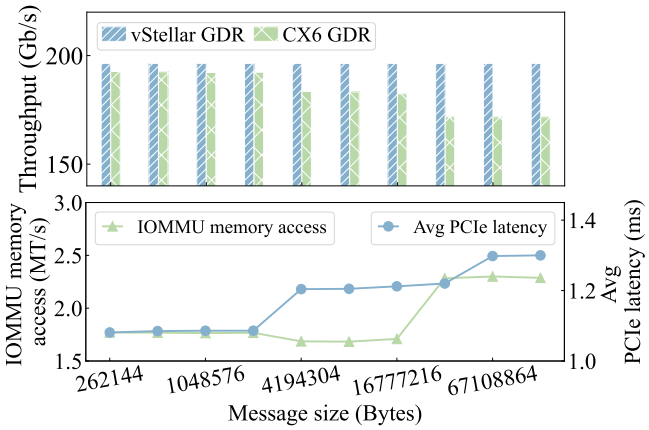


Figure 8: ATC miss testing result.

eMTT Performance. We compared vStellar with a standard PCIe ATS/ATC solution using a Mellanox ConnectX-6 200G RNIC[1]. We established 16 connections, each allocated independent GPU memory resources for transmission and reception. During the test, the client initiated GDR write operations on the 16 connections in a round-robin fashion. Each GDR write operation transferred data from local GPU memory to the server's GPU memory. The testing platform was configured with typical virtualization parameters: IOMMU=`nopt`, ATS enabled, and PCIe switch ACS DT features turned on. We set the GDR memory page size to 4 KB to create a worst-case scenario for cache performance. The results are shown in Figure 8.

For message sizes over 2 MB, the CX6's GDR performance decreased from 190 Gbps to 170 Gbps. To confirm that this performance drop was due to ATC misses, we used Neohost [25] to analyze the average latency of all PCIe operations on the client-side CX6. We observed that when the GDR performance of the CX6 decreased, the average PCIe latency increased simultaneously. Furthermore, when GDR message sizes exceeded 32 MB, the GDR performance dropped to approximately 150 Gbps. We also analyzed IOMMU memory access using Intel pcm-iio [15] and found that it had also increased, indicating that the PCIe ATS requests were further aggravating Input/Output Translation Lookaside Buffer (IOTLB) misses.

In contrast, vStellar maintained a consistent GDR bandwidth as the memory size increased, as shown in Figure 8. This demonstrates that Stellar's GDR implementation does not suffer from ATC miss issues. In summary, compared to the standard ATC-based approach, our eMTT solution not only improves GDR performance but also conserves RNIC cache resources.

7 STELLAR MULTI-PATH RDMA

Both industry and academia have proposed many load-balancing solutions to alleviate in-network hash collisions and congestion. These solutions can be classified into four categories: traffic engineering (TE), flowlet-based switching, adaptive routing (AR), and end-to-end multi-pathing. After thorough exploration and experimentation, we concluded that multi-path RDMA outperforms other approaches for large-scale AI workloads. We further tested a wide variety of multi-path algorithms and parameters, ultimately deploying a 128-path Oblivious Packet Spraying algorithm with a short Retransmission Timeout (RTO) in our production environment. In this section, we share our analysis of different load-balancing solutions and explain the rationale behind our choices. To the best of our knowledge, this is the first RDMA multi-pathing solution designed for large-scale AI workloads to be deployed in production.

7.1 Load-Balancing Approaches

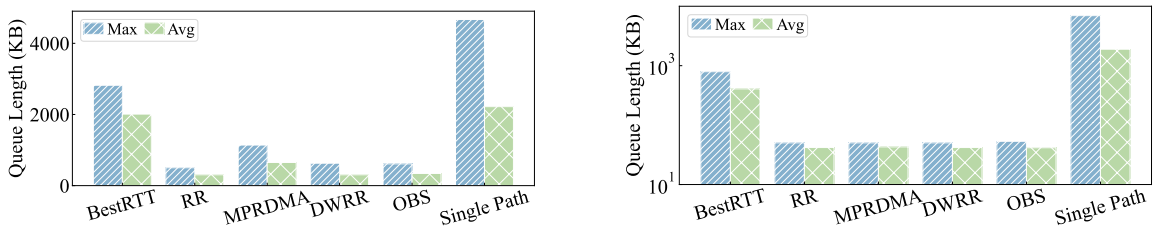
Traffic Engineering. TE solutions utilize a centralized controller to dynamically optimize routing based on real-time workloads and network topology. They leverage the regular and persistent communication patterns of LLM training tasks. As Meta has demonstrated [12], TE can outperform static ECMP and path-pinning load-balancing solutions.

However, TE is a reactive and general-purpose solution best suited for single-tenant scenarios. In a public cloud, its benefits are diminished by dynamic workloads, and it faces scalability challenges. As uncovered by Meta [12], TE also adds significant software complexity and management overhead and performs worse when multiple links fail. Furthermore, TE's effectiveness is limited by the low entropy of LLM traffic; it can only redistribute flows but cannot overcome this fundamental constraint.

Flowlet Switching. Flowlet-based switching [33] is a dynamic routing solution that detects inter-packet gaps within a flow, splits the flow into "flowlets," and routes each flowlet to a different next hop. However, as pointed out by recent studies [21, 30], flowlet-based solutions are often ineffective for RDMA load balancing due to RDMA's bulk traffic patterns. Despite this, we appreciate the simplicity and compatibility of this approach and plan to enable it in our older-generation GPU clusters.

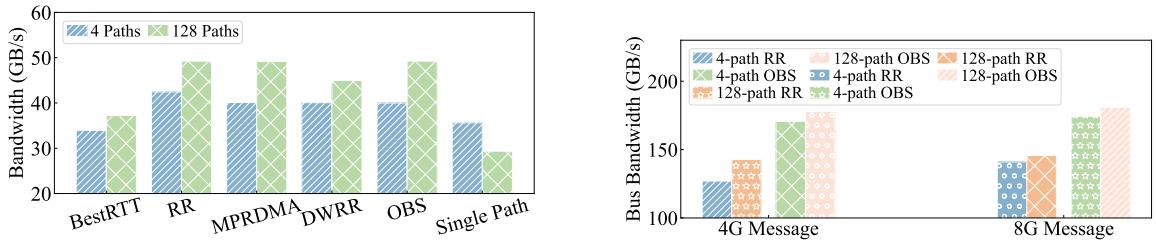
Adaptive Routing. Adaptive Routing (AR), or packet spraying, is a switch-side dynamic multi-pathing algorithm. An AR-enabled switch dynamically selects an output port for an incoming packet based on link status. Packets from the same RDMA connection can then traverse multiple paths to reach the destination RNIC, which must process the resulting out-of-order packets using technologies like Direct Packet Placement [19]. AR can achieve both

[1]Currently, ATC miss-related key counter information can only be obtained from a CX6 using Neohost [25]; this functionality is not available on the CX7.



(a) 4 paths per connection. **(b) 128 paths per connection.**

Figure 9: Queue depth for permutation traffic, comparing performance with 4 paths versus 128 paths per connection.



(a) Performance with static background traffic. **(b) Performance with bursty background traffic.**

Figure 10: AllReduce performance under different background traffic conditions: static and bursty.

coarse-grained load balancing and fine-grained congestion avoidance. However, in our case, an AR solution is inferior to an RNIC-based multi-path solution. While both approaches aim to distribute packets evenly and should offer comparable performance gains, AR introduces a significant challenge for our network monitoring and diagnostic systems, as packets with identical headers can traverse multiple different paths in a short time.

Based on the above discussion, we chose to implement RDMA multi-path load balancing on our in-house RNIC.

7.2 Multi-Path Algorithm Selection

Multi-path transport has long been studied, and many solutions have been proposed, such as MP-RDMA [21], SMaRTT-REPS [10], and STrack [17]. These solutions typically optimize for tail latency under challenging traffic patterns (e.g., skewed distributions, heavy incasts) using advanced features like per-path windowing, trimming, and SACK. However, for large-scale AI workloads, we identified different design goals.

Large-scale LLM training jobs inject balanced, regular, high-volume, and low-entropy traffic into the network [12, 27]. Our rail-optimized, dual-plane topology provides isolated and abundant paths at the ToR and aggregation layers. These large-scale, tightly-coupled, long-running training jobs are highly sensitive to performance fluctuations, and GPU clusters are prone to failures [13, 27]. The multi-path algorithm's goal is to strike a balance between all these factors to minimize communication time. More specifically, we seek to achieve three targets:

- Distribute elephant flows evenly across all available equivalent paths.
- Remain resilient to frequent link flapping and failures.
- Be simple to implement in hardware.

We conducted thorough experiments to determine the three key parameters of the Stellar multi-path algorithm: the path selection algorithm, the number of paths, and the failure detection and mitigation mechanism. Due to space limitations, we primarily present the results for the first two parameters.

Methodology. We assessed the performance of mainstream algorithms, including BestRTT, Round Robin (RR) [1], Dynamic Weighted Round-Robin (DWRR) [16], MPRDMA [22], and Oblivious Packet Spraying (OBS), using the naive single-path algorithm as our baseline. Our experiments were conducted in a cluster with an HPN7.0 [27] topology using servers with 8 GPUs and 4 RNICs each. Each RNIC has two 200 Gbps ports. The RNIC runs an in-house, window-based congestion control (CC) algorithm that adjusts based on ECN and RTT. We kept CC parameters constant across all experiments. We primarily compare the ToR switch queue length and achievable bandwidth to evaluate algorithm performance. Our current implementation relies on a Retransmission Timeout (RTO) of 250 μs to detect packet loss, a value chosen for our low-latency data center topology.

Permutation traffic. We selected 30 GPU servers from two network segments and injected permutation RDMA write traffic, creating 120 flows in total. Each RNIC sent traffic to a random destination. For each algorithm, we tested both 4-path and 128-path configurations. As illustrated in Figure 9, RR and OBS performed best with 4 paths. With 128 paths, the performance of most algorithms (excluding BestRTT and Single Path) was similar. The average and maximum queue depths decreased significantly when using 128 paths, indicating superior load balancing.

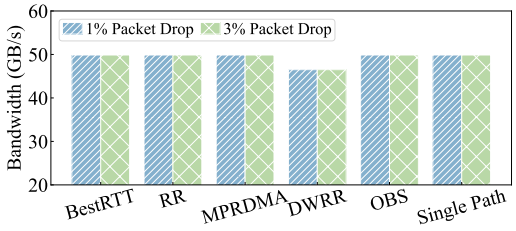


Figure 11: Performance under link failures.

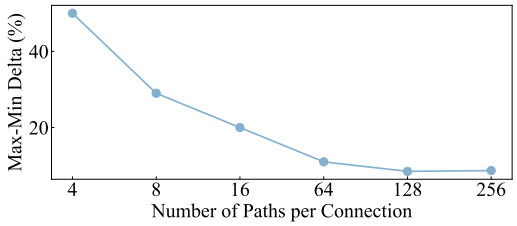
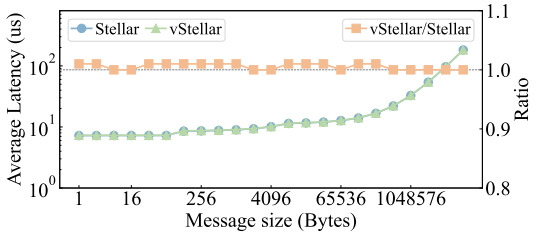
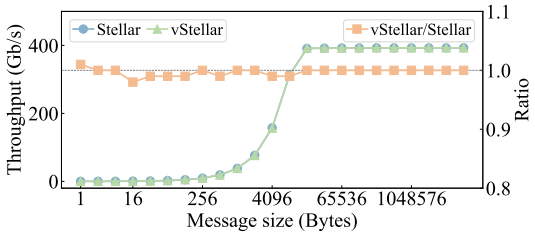


Figure 12: Evaluation of switch port load balancing.



(a) RDMA write latency.



(b) RDMA write throughput.

Figure 13: Microbenchmark results for core RDMA write operations, showing latency and throughput.

Static background traffic. We then assessed the algorithms' ability to adapt to static background traffic. We initiated two 512-GPU AllReduce tasks to simulate continuous LLM training jobs and measured the attainable bandwidth for a third 512-GPU AllReduce task. Figure 10a shows that with a sufficient number of paths (128), even simple algorithms like RR and OBS can evenly distribute traffic and fully utilize the RNIC's bandwidth (50 GB/s). In contrast, BestRTT and DWRR tended to activate only a small number of paths, leading to congestion and suboptimal performance.

Bursty background traffic. To test performance under bursty background traffic, we ran an AllReduce task that was active for 5 seconds and paused for 5 seconds cyclically. Against this persistent bursty background, we initiated a 512-GPU task using OBS and RR with 4 paths and 128 paths. Figure 10b shows the measured bus bandwidth for the test AllReduce task. Using 128 paths significantly mitigated the impact of the bursty traffic. Furthermore, OBS exhibited stronger resilience than RR, which we believe is because its pseudo-random nature interacts more favorably with our CC algorithm.

Link failures. Given that LLM training jobs are large-scale and long-running, link failures are inevitable. Our operational statistics show that a 10,000-GPU training task experiences, on average, one link failure every three days [27]. To evaluate resiliency, we initiated a 960-GPU AllReduce task and randomly dropped packets with 1% and 3% probability on a single link. As shown in Figure 11, with 128 paths, all multi-path algorithms tolerated these failures well, with almost no observable performance degradation. This is because distributing traffic over 128 paths effectively reduces the perceived packet loss rate at the endpoint by a factor of 128, making the performance loss nearly imperceptible. For complete link or optical module failures, Stellar uses a short RTO to retransmit lost packets on a different path for instant recovery. Over the long term, the control plane (*e.g.*, BGP) detects the failure and reroutes traffic, and Stellar's CC algorithm then quickly converges to a new flow-path assignment.

Path quantity. The above analysis shows that the number of paths has a substantial impact on performance. To further quantify this, we evaluated the algorithms under varying path counts by measuring the RDMA bandwidth between two RNICs with 16 connections, using 4, 8, 16, 32, 64, 128, and 256 paths. Figure 12 shows the network imbalance, calculated as the ratio of the difference between the maximum and minimum load on all ToR uplink ports to the total port bandwidth. We observed that ideal network balance was achieved only when the number of paths reached 128. This was expected, as our network's 60 aggregation switches [27] mean that 128 paths are sufficient to uniformly cover all possible routes.

In summary, the 128-path OBS algorithm not only simplifies hardware implementation but also achieves the best performance in both static and dynamic network environments, while maintaining resilience to link failures.

8 EVALUATION

8.1 vStellar's Microbenchmark Performance

We evaluated vStellar's performance in two aspects. First, we compared its RDMA benchmark performance in secure containers against bare-metal performance in regular containers, proving that the virtualization overhead of vStellar is negligible. Second, we compared vStellar with traditional GDR solutions that rely on PCIe ATS/ATC, demonstrating that vStellar has superior scalability. For these experiments, we used two GPU servers, each equipped with two Xeon CPUs, four RNICs with two 200 Gbps ports each, and eight GPUs. The servers ran CentOS 8 with OFED-23.10, and the IOMMU was configured in `nopt` mode.

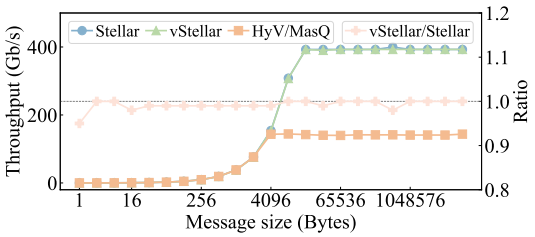


Figure 14: GDR write throughput.

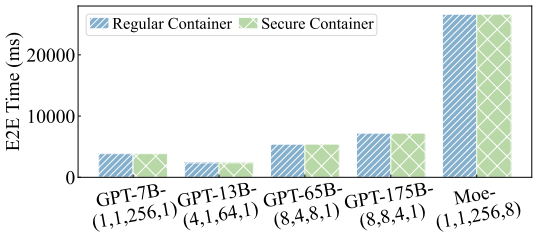
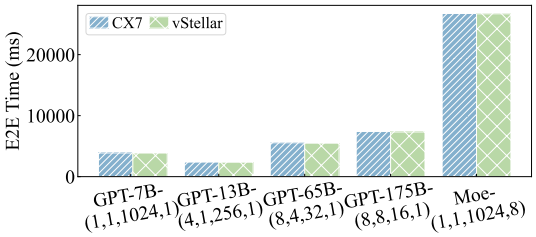
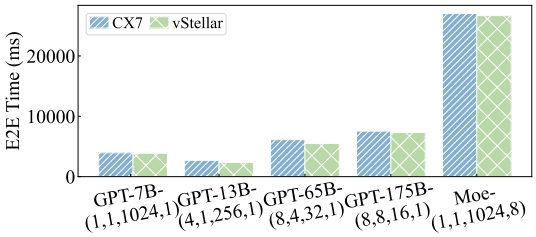


Figure 15: E2E performance with and without virtualization.



(a) With reranking.



(b) With random ranking.

Figure 16: Performance comparison with SOTA under different cluster scheduling strategies. The x-label represents TP, PP, DP, and EP, respectively.

Negligible Overhead in RDMA Performance. In this experiment, we used the 'perftest' benchmark to quantify the impact of vStellar virtualization on RDMA bandwidth and latency. To conduct a precise comparison, we tested message sizes from 2 B to 8 MB, increasing by powers of two. The client and server processes were bound to different physical CPU cores. For the RunD secure container, virtual CPUs (vCPUs) were also bound to physical cores. The results, shown in Figures 13a and 13b, indicate that the latency and throughput of vStellar and bare-metal Stellar are almost identical, suggesting that the virtualization overhead is negligible. In contrast, the competing VF+VxLAN solution on a CX7 RNIC introduces a 7% latency overhead for 8 B packets and a 9% bandwidth loss for 8 MB messages.

Scalable GDR Performance. In this experiment, we compared vStellar with both HyV/MasQ and PCIe ATS/ATC-based solutions to demonstrate its superior scalability for GDR. First, we compared the GDR performance of vStellar and HyV/MasQ. As shown in Figure 14, the maximum throughput of the HyV/MasQ GDR implementation is only 141 Gbps, approximately 36% of the maximum bandwidth of vStellar (393 Gbps). This is because neither HyV nor MasQ were optimized for GDR, causing related traffic to be routed through the PCIe RC, which severely hampers performance. Additionally, we compared the GDR performance of vStellar with that of bare-metal Stellar. The results, also shown in Figure 14, confirm that vStellar introduces no discernible overhead for GDR operations.

8.2 End-to-End LLM Training Performance

We evaluated the end-to-end training performance of large language models (LLMs) to compare our proposed Stellar framework with a state-of-the-art (SOTA) solution based on the CX7 RNIC. We conducted two separate experiments to demonstrate the benefits of our multi-path and virtualization technologies. First, we used regular (non-virtualized) containers to isolate the performance impact of the different transport-layer multi-pathing technologies. Second, we compared model training performance in regular versus secure containers to evaluate the overhead of our virtualization approach, vStellar. In this case, both container types used the same underlying Stellar transport.

Multi-path Transmission. In the first experiment, we evaluated how the transport layer affected training performance under different network congestion scenarios. We used task placement strategies—reranking and random ranking—to control the level of network congestion. Reranking is a cluster scheduling capability that co-locates communicating GPUs, keeping them physically close. This significantly reduces cross-switch traffic, thereby lowering network congestion.

Specifically, we trained models on 1,024 GPUs, with half drawn from one network segment and half from another. With reranking, which reflects a large, well-scheduled training job, network congestion was minimal, and thus the performance differences between transport layers were also minimized. The results in Figure 16a show that Stellar consistently outperforms the CX7-based solution, with an average improvement of 0.72%. Conversely, with random ranking, which simulates a cluster with many small, uncoordinated jobs, the locality of network traffic was disrupted, leading to a significant increase in network congestion. This scenario highlights the performance differences at the transport layer. As Figure 16b shows, Stellar improves training performance by an average of 6%, with a maximum increase of 14%.

Framework	Model	Parameters	TP Com.	DP Com.	PP Com.
Megatron	Llama-33B	2,3,148,1,58,8584	4.57%	20.95%	2.65%
Megatron	GPT-200B	4,12,34,1,117,3978	10.88%	1.49%	20.14%
DeepSpeed-Zero1	Llama-2B	1,1,16,1,2,32	N/A	17.3%	N/A
DeepSpeed-Zero3	Llama-13B	1,1,440,1,1,440	N/A	10.5%	N/A

Table 1: Parallel strategy and communication ratio of typical models. Numbers in the Parameters column denotes TP, PP, DP, Micro-batch Size, Gradient Accumulation, and Global-batch Size.

Virtualization. In the second experiment, we evaluated the impact of our virtualization by comparing the training performance of models running in regular versus secure containers. We used 256 GPUs, half from one network segment and half from another, and employed random ranking to create a network-intensive scenario. As shown in Figure 15, the training performance was nearly identical in both container types. This result indicates that the proposed vStellar virtualization layer does not introduce any significant performance overhead. Considering the flexibility, scalability, and security advantages brought by vStellar, it is clearly a more suitable solution for serverless AI environments.

9 DISCUSSIONS

vStellar's isolation between VMs. In vStellar, all virtual devices share the same BDF number in the PCIe subsystem. To prevent unauthorized access between VMs, vStellar implements several security mechanisms. First, it assigns standalone registers on the RNIC to each VM so that one VM cannot access another's registers. Second, vStellar leverages the "protection domain" concept from the RDMA specification. In RDMA, a queue pair can only access a memory region if both belong to the same protection domain. vStellar assigns a dedicated protection domain to each VM, ensuring that cross-domain memory accesses are rejected by the hardware. Lastly, since all RDMA resource allocation is managed by the vStellar driver, the driver can enforce additional access control and isolation policies to further enhance security.

Distributed training and computation/communication overlapping. As model sizes increase, the memory of a single GPU becomes insufficient, making parallel training necessary. Megatron-LM 3D parallelism [24] and DeepSpeed's data parallelism [28] are currently the dominant methods for LLM training. Although using many GPUs provides more aggregate computing power and memory, parallel training inevitably introduces communication overhead. Therefore, reducing this communication overhead is a core concern. As shown in Table 1, we have collected statistics for several typical parallel training jobs using Megatron-LM and DeepSpeed. In these examples, the communication-to-computation ratio ranges from 10% to 32%. While we employ computation/communication overlap techniques [18] to hide this latency, this approach does not negate the need for a high-performance network. First, communication cannot be completely overlapped in most cases. Second, implementing effective overlap requires custom adaptations to the training framework [23, 29]. Given that upper-layer application logic is still evolving rapidly (e.g., Mixture-of-Experts (MoE) introducing expert parallelism), these adaptations lag behind. During this adaptation window, which can last several months, we must rely on a high-performance network to ensure the scalability of parallel training.

Advanced multi-path algorithms. Recent multi-path algorithms such as SMaRTT-REPS [10] and STrack [17] offer superior performance under challenging traffic patterns. We implemented a similar path-aware packet spraying algorithm to test its performance in our environment. Unfortunately, we did not observe a significant performance advantage over the simpler OBS algorithm. The reason is twofold. First, the training framework and collective communication libraries already inject traffic with regular patterns (e.g., permutation) into the network. Second, our dual-plane, multi-rail topology is designed to avoid unnecessary traffic collisions. According to our operational experience, a high fan-out (128 paths) combined with a simple multi-path algorithm is sufficient to eliminate most in-network congestion for current workloads. We believe that in the near future, as training and inference workloads evolve, more challenging traffic patterns will emerge, and advanced multi-path algorithms may become necessary.

Per-path CC vs. High-fanout multi-pathing. In Stellar, all 128 paths share a single congestion control context (CCC); that is, they are controlled by a single congestion window. An alternative approach is to assign a separate CCC to each path. This would greatly reduce the total number of paths Stellar could support (from 128 down to 4) due to increased hardware resource consumption, but in exchange, it would enable a more precise, per-path congestion response. For current AI applications, which generate highly regular traffic, we believe that a higher fan-out provides greater benefits by maximizing path diversity, consistent with the results shown in Figure 12.

10 CONCLUSION

Stellar represents a significant leap forward in RDMA virtualization and multi-path transmission, specifically tailored for large-scale, cloud-based AI workloads. By systematically addressing critical limitations in existing solutions related to scalability, performance stability, and resource utilization, Stellar provides a comprehensive new framework for high-performance networking in the cloud. Our production deployment on large-scale AI clusters has demonstrated its real-world impact, delivering substantial improvements in container start-up time, virtual device scalability, and end-to-end training efficiency. The key innovations presented in this paper—PVDMA for agile memory management, eMTT for scalable GDR, and high-fan-out packet spraying for robust load balancing—provide a robust foundation for the next generation of cloud-native AI infrastructure.

ACKNOWLEDGMENT

We acknowledge all teams within Alibaba Cloud that contributed to the success of Stellar, including the High-Performance Network, Lingjun Production, Network Automation, Network Operation, Network Systems, Optical Network, and PAI teams, to name a few. We also thank our shepherd, Neil Spring, and the reviewers for their insightful comments.

REFERENCES

[1] 1991. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in communications* 9, 7 (1991), 1024–1039.

[2] 2024. NVIDIA GPUDirect RDMA. http://docs.nvidia.com/cuda/gpudirect-rdma/. (Aug 2024).

[3] 2024. PCI Express Specification. https://pcisig.com/specifications. (2024). Accessed: Sep 2024.

[4] 2024. Remote Direct Memory Accesss. (2024). https://en.wikipedia.org/wiki/Remote_direct_memory_access Accessed: Jan 2025.

[5] 2025. *Nvidia Multi-Instance GPU.* https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[6] 2025. *Nvidia vGPU.* https://docs.nvidia.com/vgpu/sizing/virtual-workstation/latest/right-gpu.html

[7] 2025. paravirtualization. https://www.techtarget.com/searchitoperations/definition/paravirtualization. (Jan 2025).

[8] E. Agostini, D. Rossetti, and S. Potluri. 2018. GPUDirect Async: Exploring GPU synchronous communication techniques for InfiniBand clusters. *J. Parallel and Distrib. Comput.* 114 (2018), 28–45. https://doi.org/10.1016/j.jpdc.2017.12.007

[9] Nadav Amit, Muli Ben-Yehuda, IBM Research, Dan Tsafrir, and Assaf Schuster. 2011. vIOMMU: Efficient IOMMU Emulation. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*. USENIX Association, Portland, OR. https://www.usenix.org/conference/usenixatc11/viommu-efficient-iommu-emulation

[10] Tommaso Bonato, Abdul Kabbani, Daniele De Sensi, Rong Pan, Yanfang Le, Costin Raiciu, Mark Handley, Timo Schneider, Nils Blach, Ahmad Ghalayini, Daniel Alves, Michael Papamichael, Adrian Caulfield, and Torsten Hoefler. 2024. FASTFLOW: Flexible Adaptive Congestion Control for High-Performance Datacenters. (2024). arXiv:cs.NI/2404.01630 https://arxiv.org/abs/2404.01630

[11] D. R. Engler, M. F. Kaashoek, and J. O'Toole. 1995. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP '95)*. Association for Computing Machinery, New York, NY, USA, 251–266. https://doi.org/10.1145/224056.224076

[12] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/3651890.3672233

[13] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. (2024). arXiv:cs.AI/2407.21783 https://arxiv.org/abs/2407.21783

[14] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. 2020. MasQ: RDMA for Virtual Private Cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York,

NY, USA, 1–14. https://doi.org/10.1145/3387514.3405849

[15] INTEL. 2022. Performance Counter Monitor. https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html. (Nov 2022).

[16] Taeck-Geun Kwon, Sook-Hyang Lee, and June-Kyung Rho. 1998. Scheduling algorithm for real-time burst traffic using dynamic weighted round robin. In *1998 IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 6. IEEE, 506–509.

[17] Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabbani, Vipin Jain, Raghava Sivaramu, and Francis Matus. 2024. STrack: A Reliable Multipath Transport for AI/ML Clusters. (2024). arXiv:cs.NI/2407.15266 https://arxiv.org/abs/2407.15266

[18] Ja-Song Leu, Dharma P Agrawal, and Jon Mauney. 1987. Modeling of parallel software for efficient computation communication overlap. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*. 569–575.

[19] Levi, Dotan David, Urman, and Avi. 2019. Direct Packet Placement. https://www.freepatentsonline.com/y2020/0092229.html. (24 Nov 2019).

[20] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 53–68. https://www.usenix.org/conference/atc22/presentation/li-zijun-rund

[21] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-path transport for RDMA in datacenters. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 357–371.

[22] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path Transport for RDMA in Datacenters. https://www.usenix.org/conference/nsdi18/presentation/lu. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 357–371.

[23] Vladimir Marjanović, Jesús Labarta, Eduard Ayguadé, and Mateo Valero. 2010. Overlapping communication and computation by using a hybrid MPI/SMPSs approach. In *Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10)*. Association for Computing Machinery, New York, NY, USA, 5–16. https://doi.org/10.1145/1810085.1810091

[24] NVIDIA. [n. d.]. Megatron-LM. https://github.com/NVIDIA/Megatron-LM/tree/main/megatron. ([n. d.]). Accessed: Sep 2024.

[25] NVIDIA. 2023. Mellanox Neo. https://docs.nvidia.com/nvidia-mellanox-neo-documentation.pdf. (Nov 2023).

[26] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R Gross. 2015. A hybrid I/O virtualization framework for RDMA-capable network interfaces. *ACM SIGPLAN Notices* 50, 7 (2015), 17–30.

[27] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 691–706. https://doi.org/10.1145/3651890.3672265

[28] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. https://doi.org/10.1145/3394486.3406703

[29] Aniruddha G Shet, Ponnuswamy Sadayappan, David E Bernholdt, Jarek Nieplocha, and Vinod Tipparaju. 2008. A framework for characterizing overlap of communication and computation in parallel applications. *Cluster Computing* 11 (2008), 75–90.

[30] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 816–831. https://doi.org/10.1145/3603269.3604849

[31] Kun Tian, Yu Zhang, Luwei Kang, Yan Zhao, and Yaozu Dong. 2020. coIOMMU: A Virtual IOMMU with Cooperative DMA Buffer Tracking for Efficient Memory Management in Direct I/O. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 479–492. https://www.usenix.org/conference/atc20/presentation/tian

[32] Michael S. Tsirkin and Cornelia Huck. 2023. Virtual I/O Device (VIRTIO) Version 1.3. https://docs.oasis-open.org/virtio/virtio/v1.3/virtio-v1.3.html. (Oct 2023).

[33] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini