**CSE 291A/DSC 291: Data Systems for Machine Learning, Spring 2026**

# 13: Sharing the GPU Communication Datapath across ML Workloads

*Lecturer: Danyang Zhuo* *Scribe:*

Nihal Nazeem, Alexiy Buynitsky, Conor Mc Gartoll, Gabrielle Kim,
Malcolm Hsiu, Ali Alabiad, Kevin Liang

## 1 Introduction

Modern ML infrastructure is increasingly bottlenecked by the communication datapath connecting compute units to memory and other GPUs, rather than raw compute capability. In this lecture, Dr. Danyang Zhuo (Assistant Professor at Duke University, previously a postdoc at UC Berkeley and PhD from University of Washington), explores how we must treat the GPU datapath (including PCIe, NVLink, and RDMA NICs) as a "first-class" shared resource. Prof. Zhuo presents a three-step research agenda spanning hardware multiplexing, workload orchestration, and accurate performance simulation to highlight his work such as Nixie, Collie/Husky, MCCS (Managed Collective Communication Service), and Phantora.

## 2 The GPU Sharing Paradigm

GPUs are highly expensive resources, and modern ML workloads are heavily memory- and compute-intensive, which implies sharing GPUs are an economic inevitability. The GPU sharing occurs in two regimes

1. **The Edge Scenario** A consumer machine with one or few GPUs run multiple models sequentially via *temporal multiplexing* (swapping between models to maximize utility)
2. **The Datacenter/Cloud Scenario** A massive cluster of data center GPUs (e.g A100, H100, B200) managed by orchestrators like Slum or Kubernetes, where multiple tenants arrive and leave, running diverse training, inference, and RL jobs using both *temporal and spatial multiplexing*

### 2.1 The Datapath Bottleneck

Sharing a GPU means sharing a datapath between the GPU and the rest of the system.

- **Edge Level** The PCIe bus handles interactions betwene GPU and host/CPU memory
- **Inter-GPU (Intra-node)** NVLink, NVLink C2C, and TPU InterCore Interconnects (ICI) handle high-bandwdth communication between accelerators
- **Datacenter Level (Inter-node)** RDMA (Infiniband, RoCE) over Ethernet connects servers

Prof. Zhuo proposes a three-step research approach to optimize shared GPU computing:

1. Ensure the underlying hardware supports secure and efficient multiplexing
2. Orchestrate workloads to be resource-efficient, maximizing network bandwidth on shared GPU fabrics
3. Allow applications to reconfigure themselves with confidence to maximize MFU (Model FLOPs Utilization)

# 3 Step 1: Resource-Efficient Edge Workload Orchestration (Nixie)

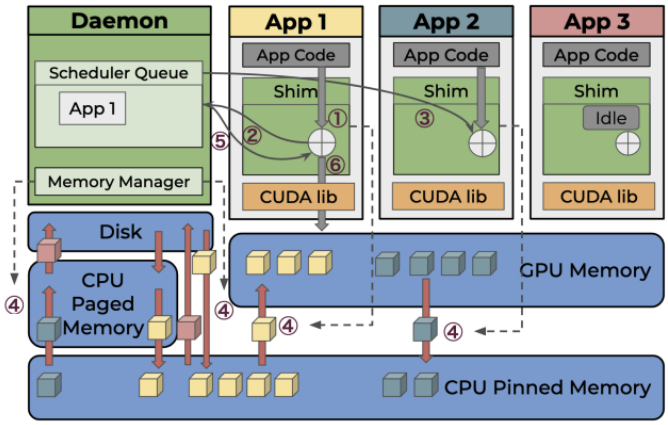


Figure 1: *Nixie Architecture: green sections are owned by Nixie while other colors denote other application memory. Figure directly from Nixie paper.*

Consider an edge workflow where an LLM is used to describe a scene and a diffusion model uses the text to generate an image. The user than queries the LLM to verify the image quality in this loop.

If Model A (LLM) and Model B (Diffusion) each require 30GB of VRAM, but the system has a single 32GB GPU, the user faces a trade-off:

1. Compress both models to 10GB each and run them simultaneously (sacrificing quality)
2. Run the full 30GB models via temporal multiplexing, swapping their weights in and out of the GPU

Deploying models on the edge requires one or a few GPUs. This calls for far simpler resource management when contrasted with the complicated workflows and runs executed on hundreds or even thousands of GPUs in a datacenter.

Additionally, edge deployments, are configured to match the target hardware (e.g. ensuring multiplexing support). Furthermore, edge deployments don't manage trying multiple communication topologies due to low device count. As a result, optimization efforts for edge workloads are focused on ensuring resource-efficient workload orchestration.

### 3.1 Challenges with Edge Workloads (e.g. NVIDIA Unified Memory)

If we reply on standard OS paging via the NVIDIA Unified Memory (UVM) for temporal multiplexing, the system

Deploying on edge frequently faces challenges around fixed compute bandwidth that cannot be resolved simply by adding more compute. If we reply on standard OS paging via the NVIDIA Unified Memory (UVM) for temporal multiplexing, the system encounters the following issues:

- **Excessive Pinned Memory**: CPU is required to pin (allocate) memory for all models currently in GPU HBM memory. If we have two models, Model A (30GB) and Model B(30GB), the CPU will pin 60GB of RAM / SSD to hold model parameters. In the case, that one (or both) of the models get evicted from GPU
- **Trashing**: Model A and Model B compete for GPU resources (mostly memory), leading to degraded performance for both models if the working set is the same as the GPU size and they evict each other.
- **Poor PCIe Bandwidth Utilization**: Although PCIe is designed to support full-duplex links (independent send and receive paths), most workloads tends to utilize only one direction (e.g. from source to destination: evicting A to CPU then loading B to GPU). As a result the opposite path (e.g. from destination to source) remains significantly underutilized.

### 3.2 The Nixie Solution

Nixie was built to address the challenges listed above by enabling efficient temporal multiplexing at the edge by managing PCIe bandwidth as a shared resource. Importantly, Nixie ensures:

1. **Efficient memory utilization.** Each chunk of data only has 1 copy, reducing redundant and wasteful data storage
2. **Coordinates compute and memory management.** When model A is running, A can evict other models from memory, not vice versa.
3. **Leverages full duplex PCIe bandwidth.** While transfering data from GPU to pinned CPU memory, Nixie allows data to flow in the complementary direction, allows data chunks to move through all memory hierarchies, and utilizing the full PCIe bandwidth.

This allows temporal switching between massive models to be nearly instantaneous, without requiring application-level reconfiguration.

## 4 Step 2: Hardware Multiplexing in the Datacenter (Collie & Husky)

A datacenter has many GPUs as well as a fast datacenter network connecting these GPUs. Within a rack, GPUs may be connected by NVLink or NVSwitch, and on a coarser granularity, they may be connected by Ethernet (running RDMA and RoCE).

In the cloud (Azure, AWS, CoreWeave, etc.) , multiple tenants share a datacenter network in order to achieve high hardware utilization. The security and performance question being investigated: *Can one tenant destroy another tenants network performance?*

While network switches can enforce rate limits (e.g. 100 Gbps per tenant), GPU-Direct RDMA relies on Network Interface Cards (NICs) that bypass the CPU. RDMA NICs manage complex internal states, including address translation tables and send/receive buffers.

### 4.1 Finding the "Narrow Waist"

Prof's Zhuo's research identifies 16 hidden hardware resources inside RDMA NICs shared across tenants. If an attacker (or buggy application) exhausts just one of these resources, they can deny network access to other tenants and degrade ML training performance.

Since standard benchmarks don't stress the hidden states, the team developed a search harness:

1. **Define the Search Space** over possible workloads on a RDMA system. The search space is restricted to a subset, called the "narrow waist", of RDMA functionality. The RDMA verb abstractions are parametrized (e.g. queue depths, memory region patterns)

2. **Use Hardware Counters as Signals** Monitor diagnostic counters (like cache miss rates or PCIe backpressure) on the NIC. The lower/higher the counter is, the more likely the test case will trigger an anomaly.

3. **Mutate Workloads** An agent/test harness proposes new workloads, mutating the workloads in the direction that increases the diagnostic counters until the performance collapses.

This lead to Collie and Husky discovering various (7) hardware vulnerabilities that were reported to NVIDIA as Common Vulnerabilities and exposures (CVEs).

## 5 Step 3: Shared Datacenter Fabric Orchestration (MCSS)

### 5.1 Challenges with Collective Communication for Datacenters

- **Collective Communication Algorithms**: Within a datacenter, there are a variety of methods for using collective communication (all-gather, all-reduce, broadcast etc.) for intra-rack and inter-rack communication (with intra-rack communication having higher bandwidth). In most ideal situations, you maximize intra-rack communication and reduce inter-rack communication, although this is very difficult to ensure traditionally in large clouds where knowledge of node locations is abstracted from the client as the clients/tenants operate inside Virtual Machines (VMs).

- **Broadcast as a Motivating Example**: Broadcast is a useful collective operation—all-gather is a variant of broadcast, and in reinforcement learning, the trainer broadcasts model weights to a set of inference servers for rollout. Consider broadcasting a buffer held at node A to nodes B, C, and D. There are multiple possible broadcast trees (options 1, 2, 3). Options 2 or 3 may be preferable since cross-rack links have less bandwidth than within-rack links. However, in the cloud, tenants only have the abstraction of virtual machines and do not know where their nodes are physically located in the datacenter network. This makes it very difficult to select the right collective communication algorithm, and picking the wrong one can result in terrible performance.

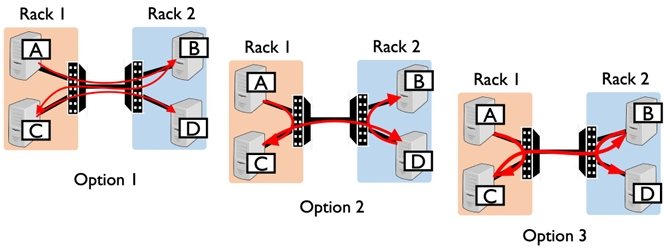


Figure 2: *Choosing a Collective Communication Algorithm*

## 5.2 Managed Collective Communication as a Service (MCCS)

Instead of a typical collective communication, MCCS is a library linked to the application; MCCS proposes treating collective communication as a managed cloud network service rather than a client-side library (e.g. linked NCCL library).

- **Network Configuration Visibility**: Since the cloud provider themselves manages the collective communication, all job orchestration can now have a detailed understanding of the specific nodes, GPUs, and servers being used and optimize the collective communication accordingly.
- **Multi-Tenant Visibility**: MCCS can also observe network usage across multiple clients and jobs. This enables the cloud provider to reason about shared fabric contention and avoid communication schedules that overload the same network links. In contrast, individual tenants typically only see their own job and cannot account for congestion caused by other workloads.

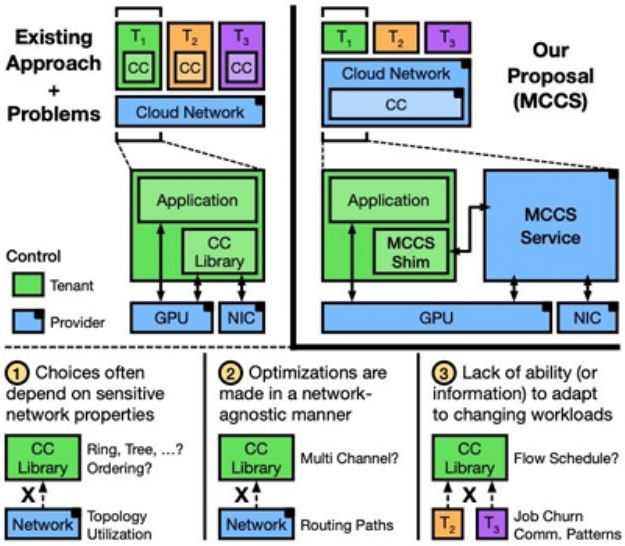


Figure 3: *Comparison between existing approaches and MCCS. Figure from the MCCS paper.*

# 6 Step 4: Reconfiguring with Confidence (Phantora)

Modern ML training requires tuning complex configurations (3D parallelism dimensions, rematerialization strategies, etc.). Before deploying a run across thousands of GPUs, engineers need to accurately predict the MFU.

## 6.1 The Limitations of Existing Predictors

We note the following limitations:

- **Mathematical Cost Models** Previous approaches modeled performance as a function of hardware and workload configuration, but this is hard to generalize for arbitrary PyTorch and dynamic features. And different frameworks will have different definitions for things like parallelism that you have to account for.
- **Trace-Based Simulation** If you trace a workload on 64 GPUs, can't extrapolate it to 128 GPUs since the trace lacks the point-to-point communication semantics between node 64 and node 66.

## 6.2 Phantora: Unmodified Framework Solution

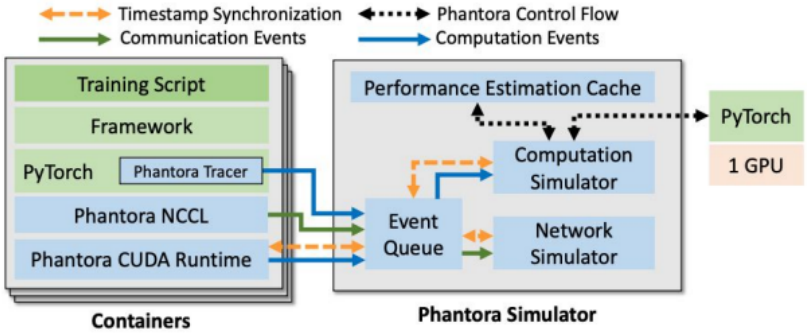


Figure 4: *Overview of the Phantora simulator.*

As shown in Figure 4, Phantora runs the unmodified training frameworks in a simulator. It interposes on how PyTorch invokes the CUDA caching allocator `cudaMalloc`, `cudaFree`, and NCCL. To simulate a 1,000-GPU cluster, Phantora runs 1,00 Docker containers executing the standard Python framework code, profile the runtime of a specific operator on one representative GPU, and when that operator is invoked in simulation, Phantora advances the virtual clock of the corresponding CUDA stream by the profiled duration. For collective communications (e.g. $n$-node Ring All-Reduce), Phantora simulates the network transfer and synchronizes the virtual clocks across all participating ranks.

Result: Phantora can predict the MFU for LLaMa-3 8B on 128 H100 in one minute using one GPU, matching Meta's publicly reported benchmarks.

Another key advantage is that framework-specific features run without any modification. For example, Megatron-LM's selective activation recomputation: activations are thrown away at runtime during the forward pass and recomputed during the backward pass. The effect of turning this feature on versus off can be observed across different configurations. The same applies to gradient accumulation. To run these features in the simulator, you just turn it on in torch as if you have a real GPU cluster.

# 7 Q&A and Discussion

## 7.1 "Cold Start" and MCCS

Q: Does MCCS allow for warm-starting the collective topology, given NCCL faces "Cold Start" latency of building optimal communication rings (say can a new job inherit a pre-optimized known communication tree from the cloud provider, or does the MCCS layer still have to re-probe the network for each new tenant?)

A: Since the Cloud Provider would know their network situation the best, and the consumer doesn't know the network topology or links, the Cloud Provider can read link/switch counters to know how the network is used. It's hard for the tenant to select it; maybe need to do probing, but if the cloud can monitor these all the time there shouldn't be a need to incur additional probing to get network conditions and give a communication strategy (Ring, Tree All-Reduce or how it's being constructed)

## 7.2 Multiple GPU Simulation

Q: Say the topology contains various kinds of GPUs, is one representative GPU still used to simulate the workload MFU?

A: A representative GPU for each of the various kinds of GPUs would be used to simulate the workloads assigned to them, other wise advance the virtual clock would be undefined.

## 7.3 Fault Tolerance and Elasticity

Q: What happens when GPUs go offline in large clusters?

A: Handling node failures requires either frequent check pointing and restarting or elastic training. Elasticity is somewhat solvable for Data Parallelism, but an active area of research for Tensor Parallelism due to sharded model weights. Current most frontier labs rely on check pointing and hibernating jobs rather than dynamically reconfiguring sharded states.

- Google's TPU Approach: Google utilizes Optical Circuit Switches (OCS) to dynamically reconfigure the 3D Torus topology when a node fails, effectively healing the network at the hardware level. Conversely, NVIDIA relies more on software and switches (NVLink/Infiniband) to handle routing.

## 7.4 Hardware and Networking Strategies

Q: How do Chinese labs (e.g. Huawei) compensate for weaker compute due to export controls?

A: Because ML jobs are often bandwidth-constrained, investing heavily in interconnects can offset weaker FLOPs. Labs using Huawei chips compensate for the lack of NVIDIA hardware by heavily investing in CUDA-equivalent kernels and highly efficient network topolgies.

- Future Networking Trends: The community is moving toward Multipath Reliable Transport (e.g. recent protocols proposed by OpenAI) to leverage all possible point-to-point links for massive GPU transfers, moving away from rigid OS-level networking.

## 7.5 Data Corruption and Security

Q: Can data be corrupted or sniffer over the network?

A: DRAM is a physical device susceptible to bitflips (e.g Rowhammer). Data corruption is a massive concern at scale, requiring robust ECC. For security NVIDA offers Confidential Computing (enclaves), which encrypts memory before transmission. But this introduces significant performance overhead, creating a tradeoff between strict security and maximum throughput.

## 7.6 Nixie Transparency and Practicality

Q: How practical is Nixie in a real desktop setup? Does Nixie need to control the entire workflow to know when Model A or Model B will run? Can it work across different applications?

A: Nixie is fully transparent. It uses `LD_PRELOAD` to extract all CUDA kernel launch operations. There is a lot of evaluation in the paper, such as diffusion models and applications like ComfyUI. It interposes at the CUDA kernel launch API, like `malloc` and `free`—quite low-level APIs. Nixie is application-agnostic: it does not know what the application is doing, whether it is an inference engine, training system, or anything else, and views it as a black-box CUDA application.

## 7.7 Evolution of Networking Research (2020–2026)

Q: Between 2020 and 2026 the world changed so much—what is the difference now in the networking community?

A: At different times, people care about different aspects of networking. In the current age, people care about two things or a few things. The first is GPU-to-GPU transfer at the end-host level—specifically GPU Direct RDMA, with high bandwidth and high throughput transfer. The second is multipath capability at the datacenter network level. There is path-level redundancy between two nodes due to many redundant paths, which is good for fault tolerance. But because transfer sizes are now large, the goal is to leverage all possible paths for point-to-point transfer. Just several days ago, OpenAI proposed a new transport called Multipath Reliable Transport, with ongoing work with hardware vendors to standardize this as a Multipath Reliable Protocol. Overall, reliability, correctness, and performance are the current focus of the networking community in GPU clusters.