
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Operationalizing High-Performance GPU Clusters in Kubernetes: Lessons Learned from Training Databricks DBRX

Will Gleich & Wai Wu
Databricks
November 13, 2024

Introductions

Will Gleich (he/him/his)

- DevOps Engineer
- Focus on Kubernetes Infra and MLOps

databricks mosaic research

Wai Wu (he/him/his)

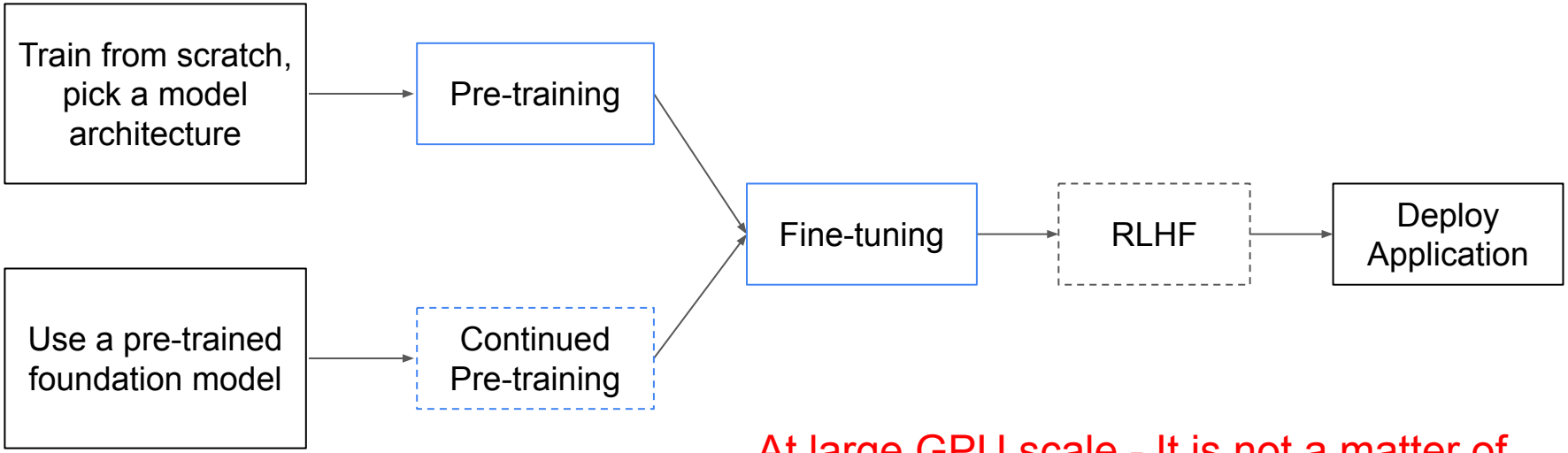
- Software Engineer
- Focus on Training

Talk Overview

- Examples of generative AI training use cases requiring large clusters
- What makes GPU clusters different from CPU clusters
- How to detect GPU failures with passive monitoring with example failure cases
- Overview of GDRDMA for generative AI training
- Active health checks and auto-mitigation design

Generative AI Training

Fault tolerance is important to power pretraining and finetuning use cases on the Databricks Data Intelligence Platform



At large GPU scale - It is not a matter of if, but when a failure will happen

DBRX Training Scale

Scale of workloads (Pretraining)

Number of Parameters: 132B parameters; behaves like 36B parameters (Mixture of Experts)

Data size: 12T tokens

Model size: 132B parameters, 36B active parameters

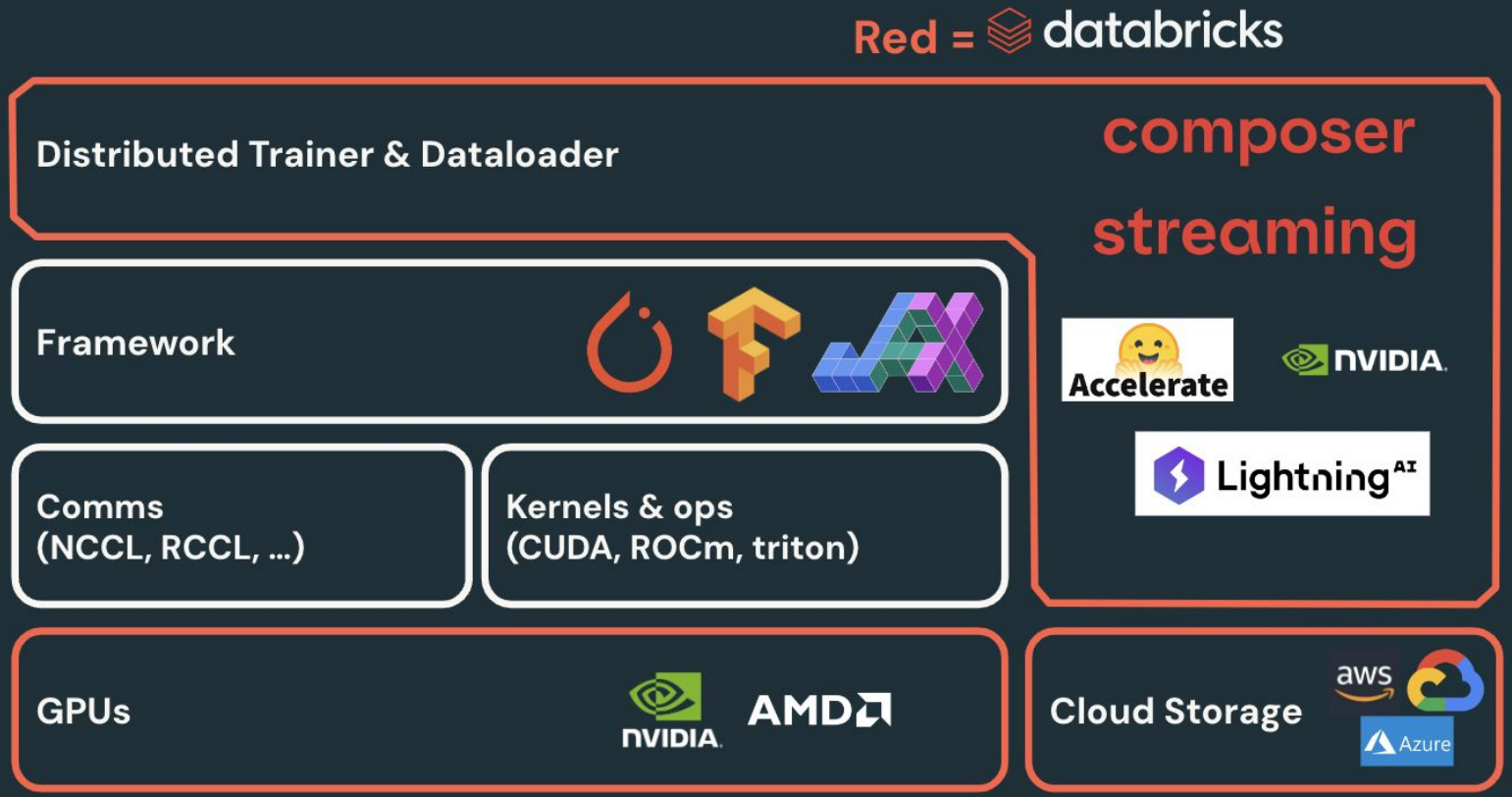
Workload size: 3072 NVIDIA H100 GPUs (384 nodes!)

Training Time: 2.5 Months

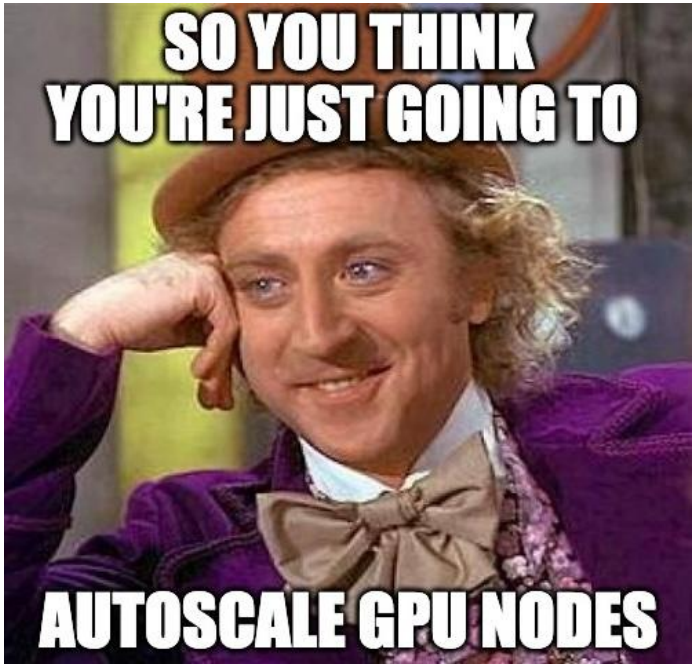
Leading model as of March 2024 release

D B
R X

Deep Learning Stack



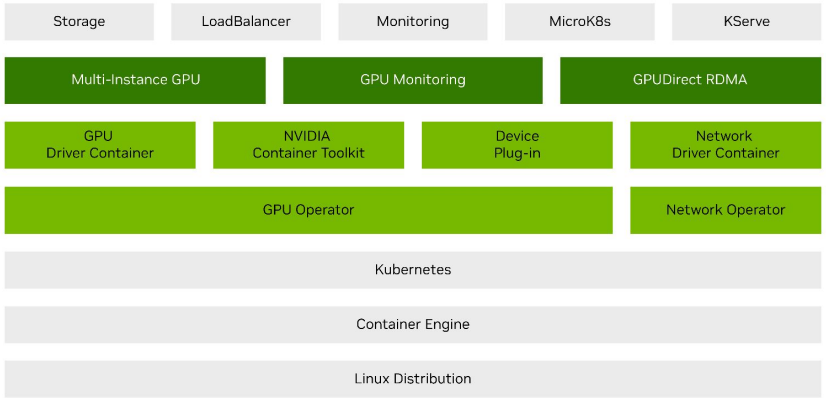
What Makes GPU Clusters Different from CPU Clusters?



- GPU instances are contended in clouds, and difficult to obtain through on-demand and auto scaling requests
  - Cloud service providers often require additional contracts around purchasing (e.g. 3 year contract)
- Additional hardware, drivers, monitors, toolkits and plugins increase the complexity and software surface of deployment
- Failures are also difficult to detect by cloud providers, nodes with bad hardware need to be reported via support requests - not just terminated

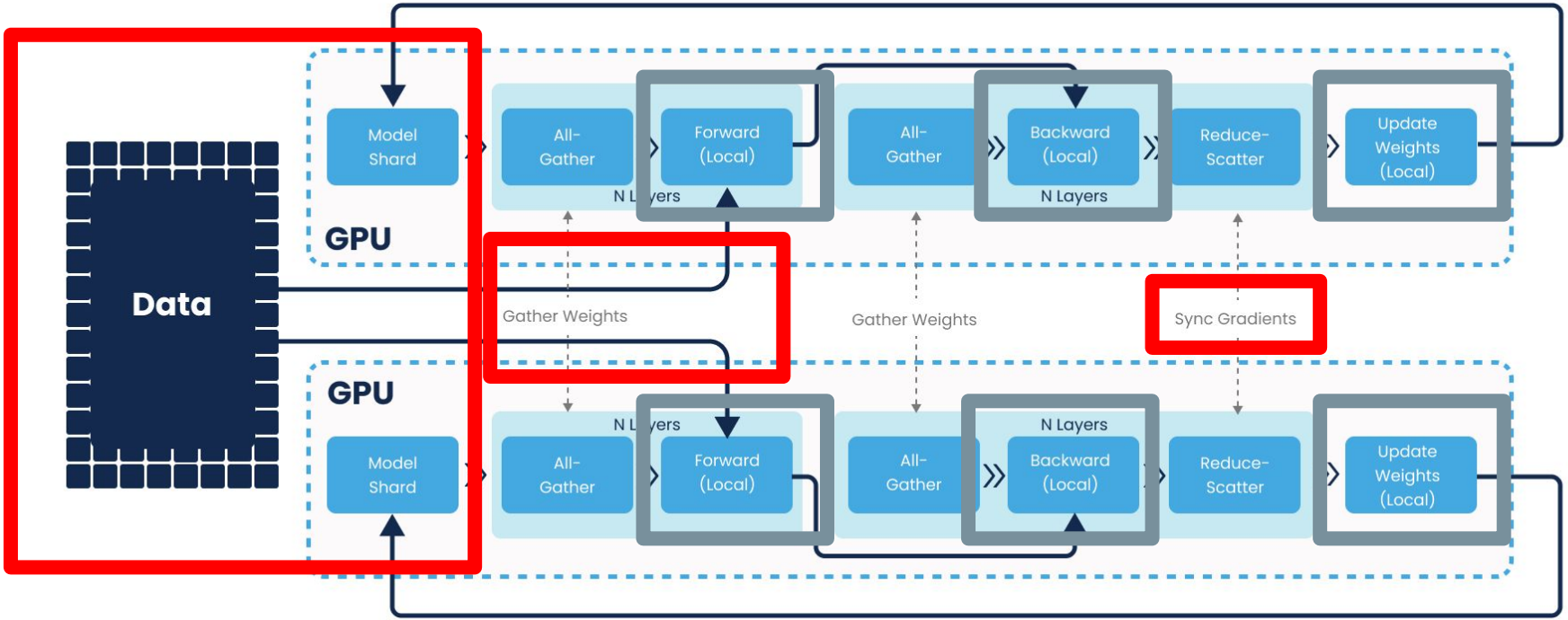
Driver Installation

- NVIDIA offers the GPU Operator, a Kubernetes Operator, that allows for easy deployment of NVIDIA drivers on nodes that are configured for GPU
  - Shown to the right is the containerized GPU Operator Diagram
- Configurability - enabling and disabling of auto-update of GPU driver across the entire cluster for easy change management
  - Be sure to test this functionality
- Drivers can also be configured on the Node OS Image with the NVIDIA Driver installed



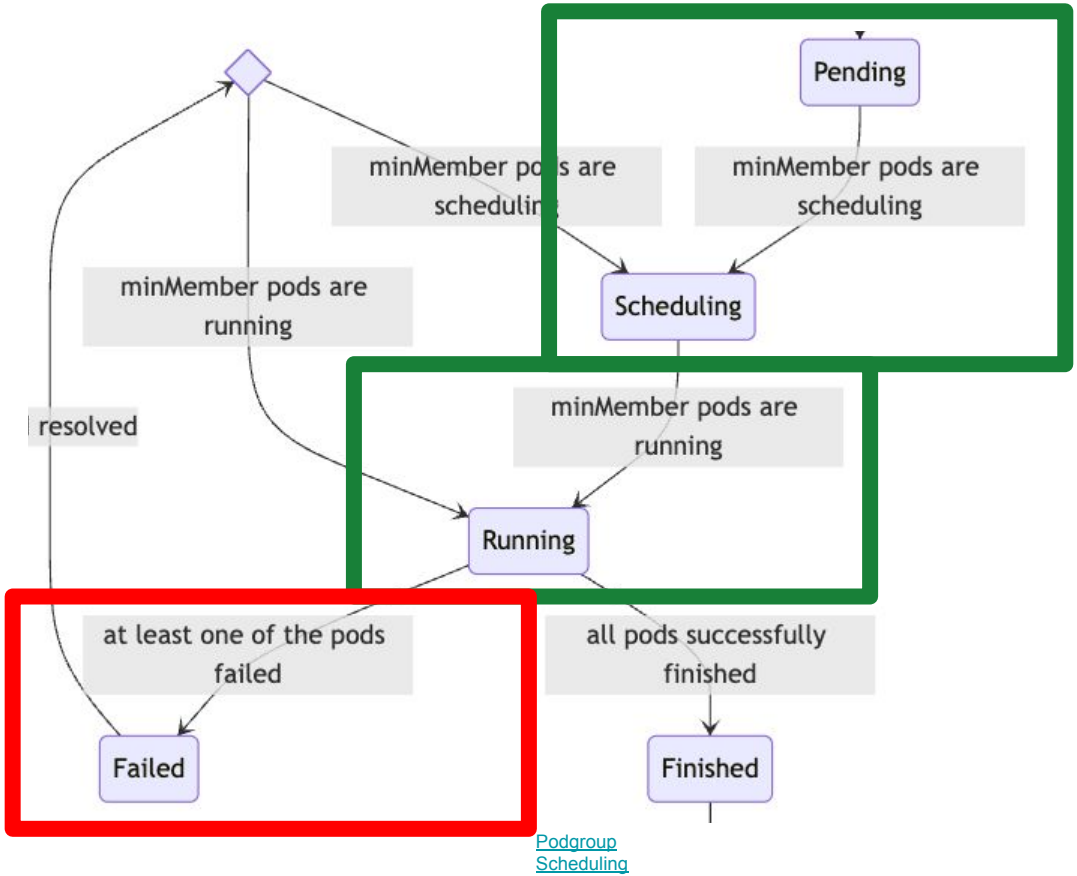
Large Language Model Training Overview

- Data and model is split between GPUs across nodes
- Partitioning allows each GPU to handle fraction of workload
- However, N-1 processes can timeout waiting for 1 hung process



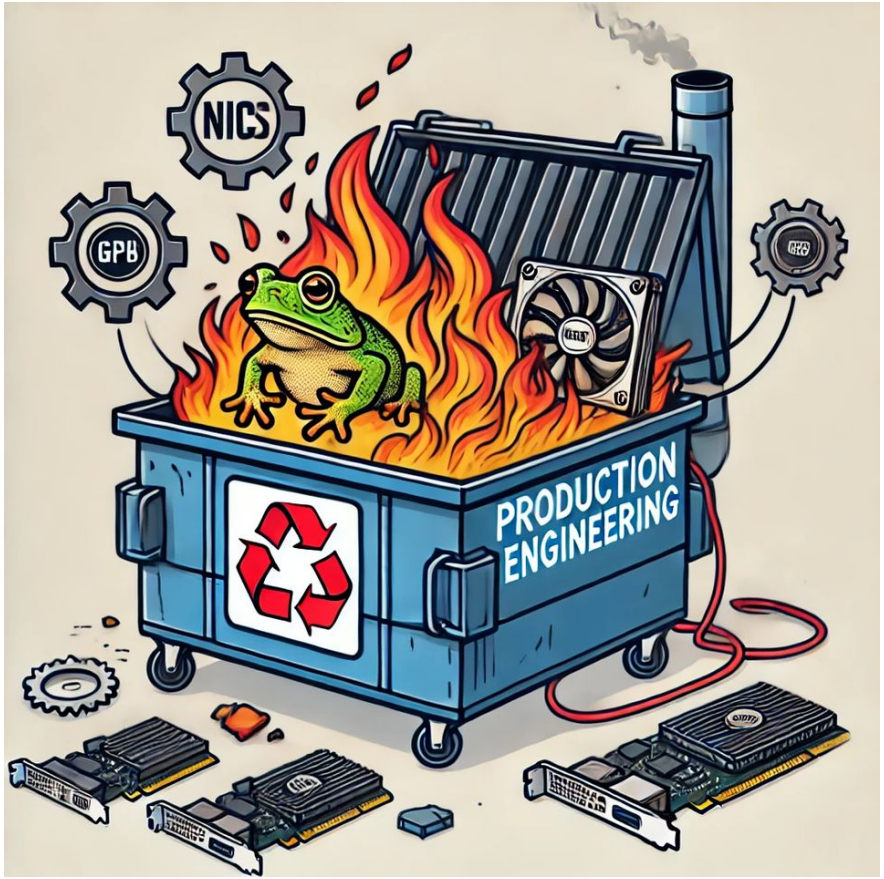
Failures Are Amplified w/ Gang Scheduling

- Pods are gang scheduled on nodes
  - via podgroups (link)
- A single node failure forces restart (from last checkpoint)
  - Higher probability of placement on >= 1 unhealthy node at scale
  - Gang scheduling causes good nodes to sit idle



Many Root Causes for Faulty Node

- Xid Errors (Over 140 kinds)
  - NVLink error
  - GPU fall off the bus
  - ECC error (data corruption in memory)
  - PCIe bus error
- GPU thermal degradation
- Connect-X NIC failure
- Filesystem failure
- Kubelet / Node CPU Failure



Monitoring via NVIDIA DCGM and Prometheus


```
Xid (PCI:0000:2d:00): 74, pid='<unknown>', name=<unknown>, NVLink: fatal error detected on link 10
```

- **Logs:** NVIDIA GPUs offer kernel logs for known GPU Faults
  - Available in the DMESG Log
- **Metrics:** Errors can be exported by NVIDIA Data Center GPU Manager (DCGM) as Prometheus metrics
  - DCGM_EXP_XID_ERRORS_COUNT
  - DCGM_FI_DEV_THERMAL_VIOLATION
  - ECC uncorrectable errors and memory remapping errors
- **Alert:** Alerting on violations to quickly detect and respond to issues

NVIDIA DCGM

Prometheus

ThermalViolationDetected (15 active)

```
name: ThermalViolationDetected
expr: sum without (exported_pod, exported_container, exported_namespace, UUID, instance) (DCGM_FI_DEV_THERMAL_VIOLATION) > 0
for: 30s
labels:
   severity: infra-low
annotations:
   summary: Thermal violation detected on GPU
```

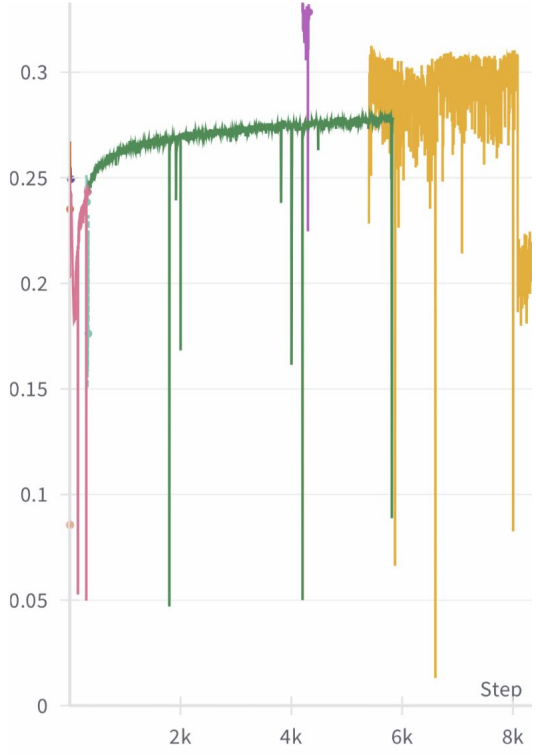
Labels	State	Active Since	Value
DCGM_FI_DRIVER_VERSION=535.86.10 Hostname=nvidia-dcgm-exporter-qgcd6 alertname=ThermalViolationDetected container=nvidia-dcgm-exporter device=nvidia0	FIRING	2024-11-09T00:30:02.96776133Z	1583898880

DCGM Exporter Configuration

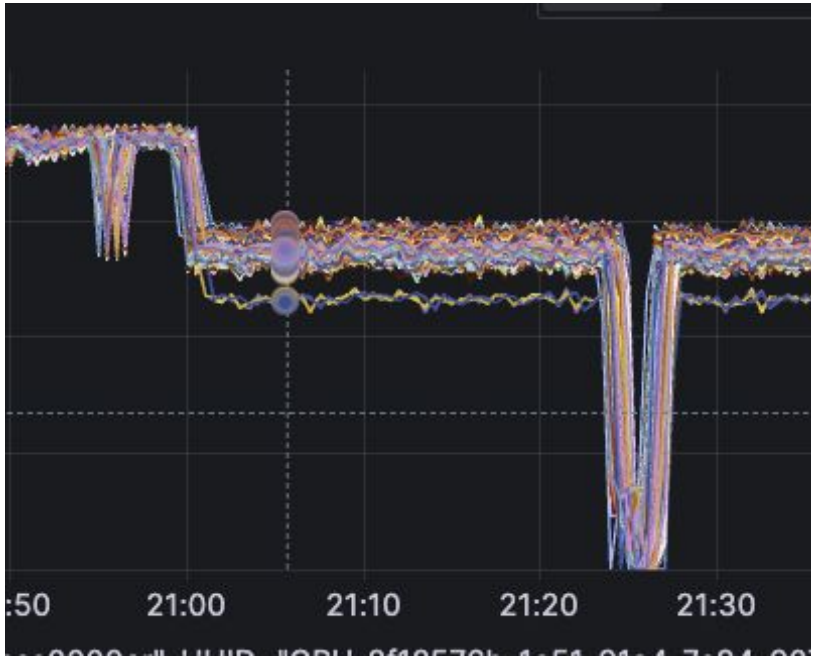
- Default config of NVIDIA DCGM Exporter omits error counters and violations
- Leverage the customized metrics options of the GPU Operator

Detect Faulty Nodes with Passive Monitoring

Run metrics - MFU Drop



System metrics - PCIE_TX_BYTE



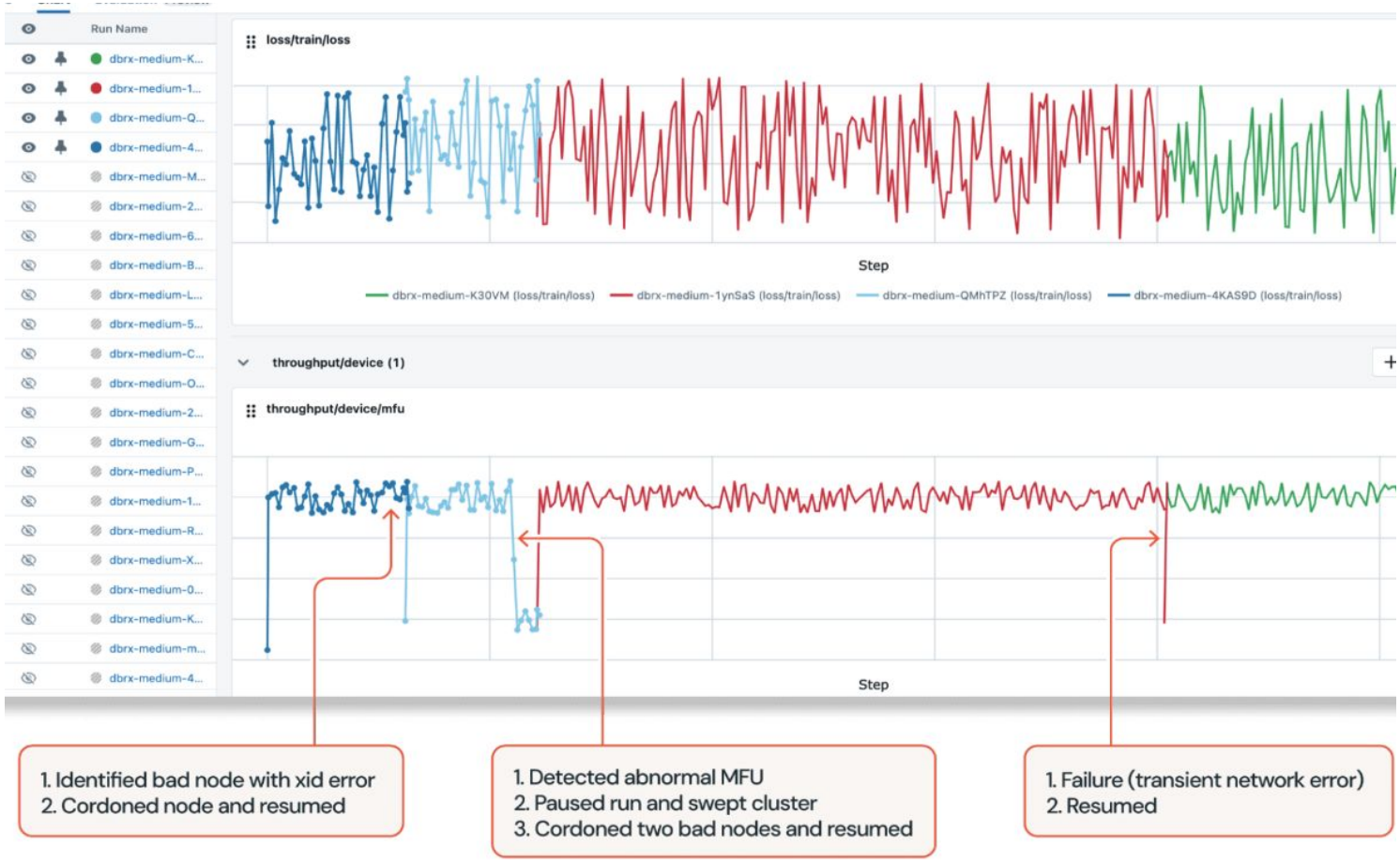
Observe 30% drop MFU ~ 10% reduction in PCIe traffic from a node

Limitations of Passive Monitoring

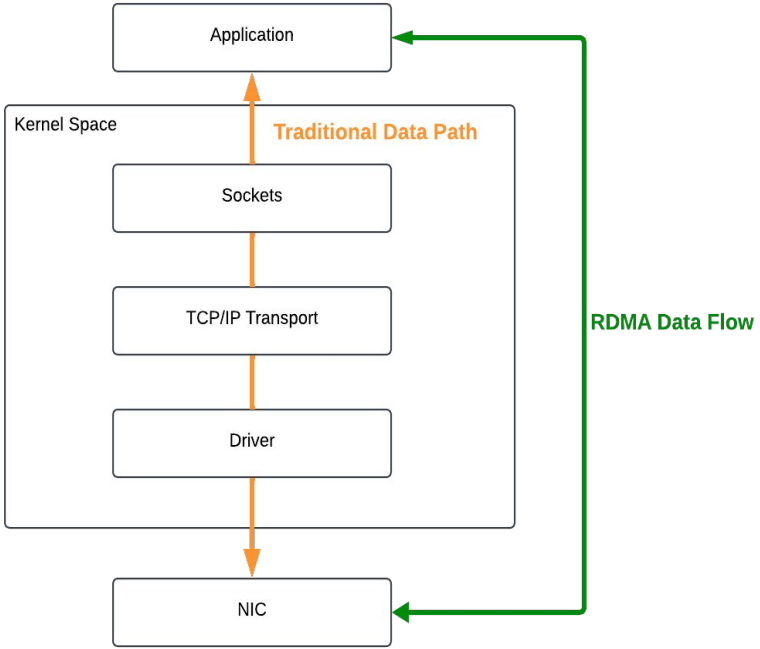
- Xid monitoring have narrow event scope
- There are unattributed hardware errors
- Need for automated diagnostic w/ run-level metrics (MFU)
  - Platform detects runs w/ low MFU
  - Sweeps w/ smaller runs
  - Cordon nodes
  - Auto resume workload from the latest checkpoint

Auto-detect Node Stragglers

mlflow

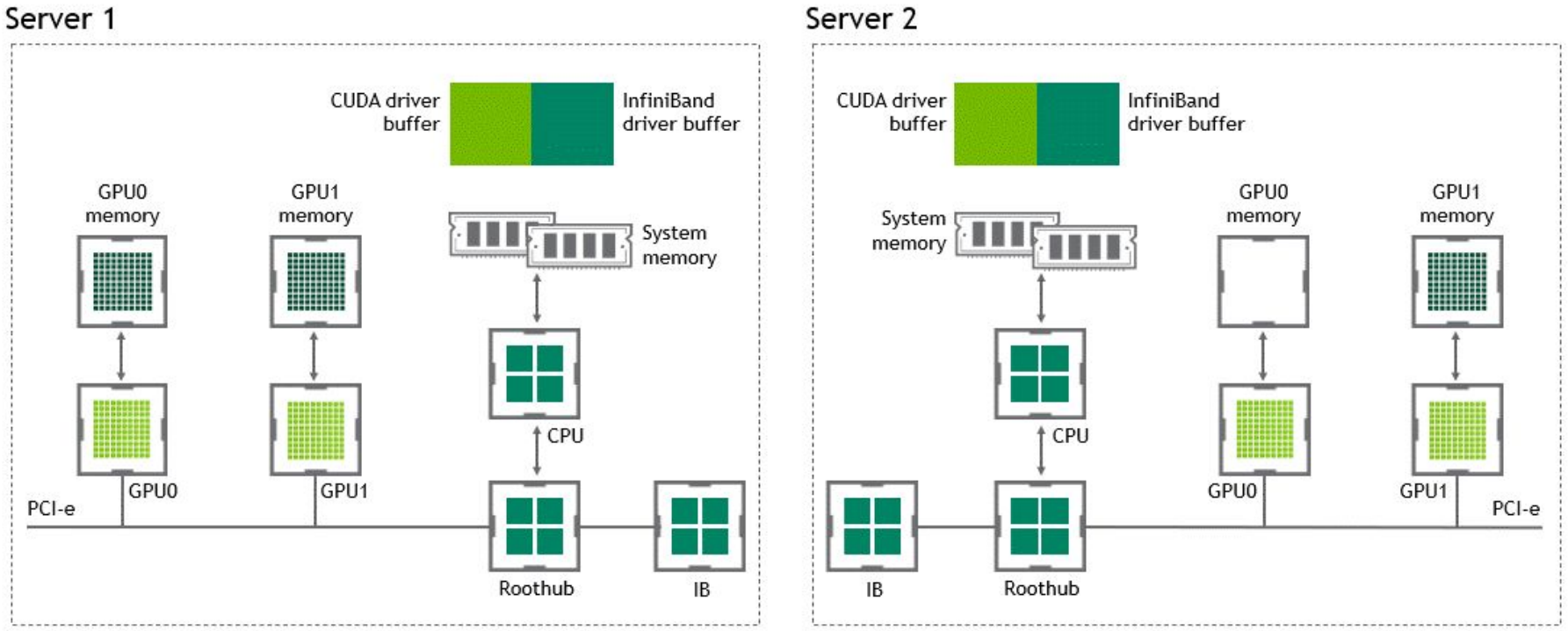


RDMA Overview

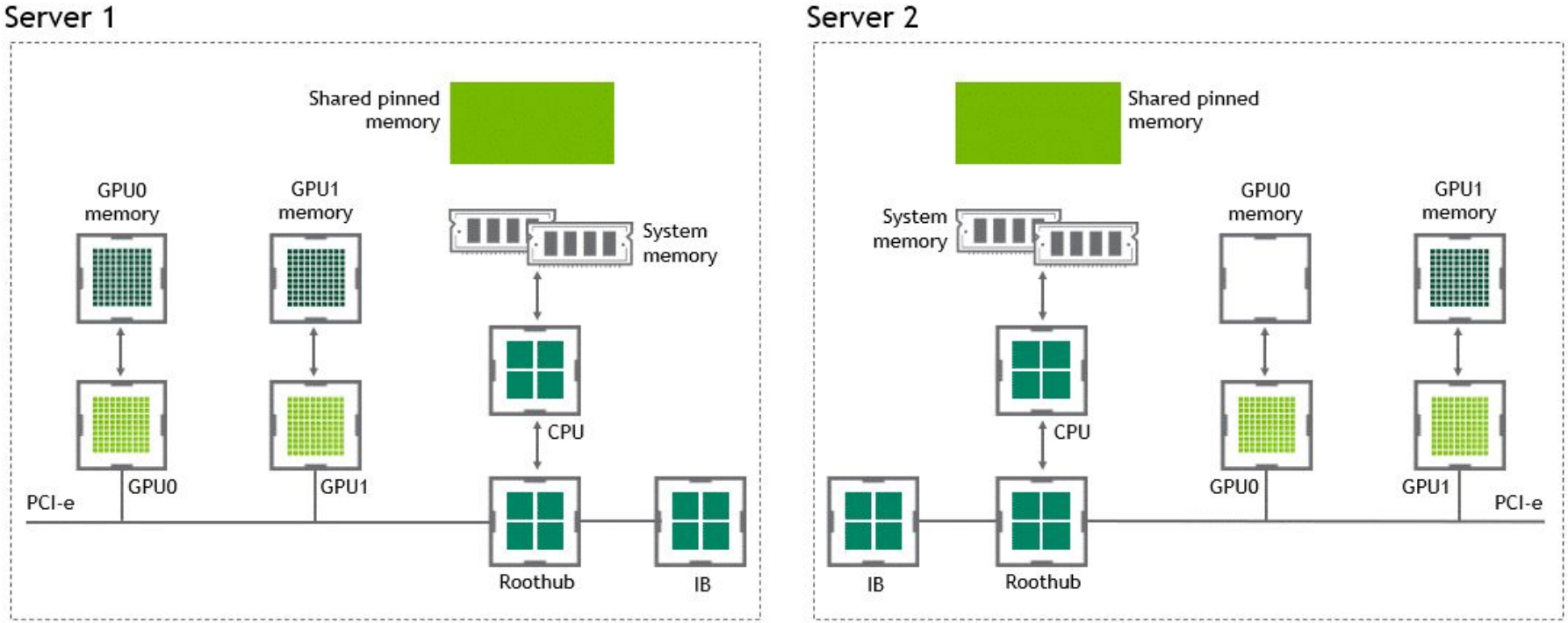


- GPUDirect Remote Direct Memory Access (GDRDMA/RDMA) provides direct communication between GPU memory in remote systems eliminating the need for CPU and system memory buffering
- This is a core technical requirement for large model fine tuning as well as foundational model pretraining

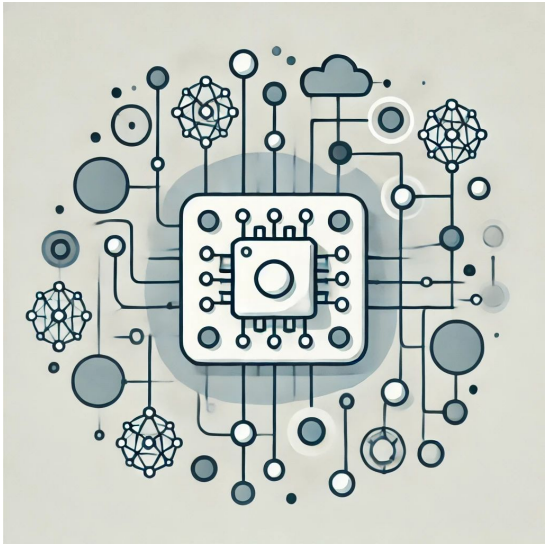
Multi Node GPU Memory Path



Multi Node GPU Memory Path GDRDMA



RDMA Monitoring Challenges



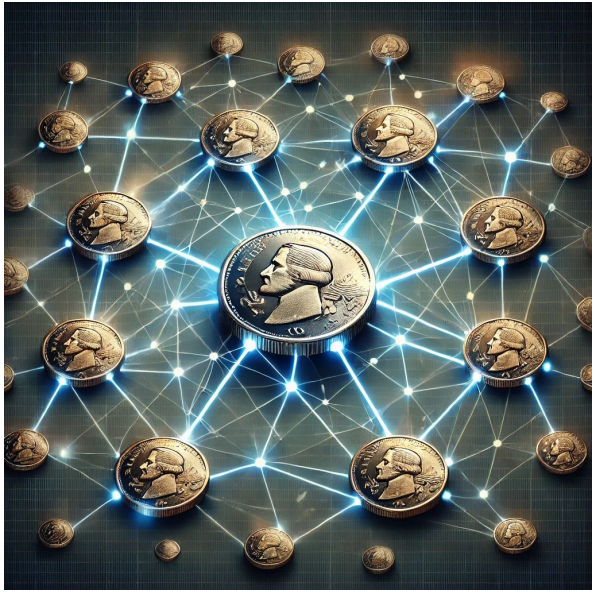
- RDMA traffic bypasses the CPU entirely - this interaction poses additional challenges for monitoring
  - Traditional node and networking monitoring don’t have visibility into the network performance
- Some performance insights can be gleaned from PCIe and NVLINK system metrics
  - Collective operations slow down the whole network of GPU workers in the workload
- Multi node performance can only be monitored on the application managing the GPUs or on the network switch

GDRDMA Cloud Networking Configuration

Cloud service providers offer different options when it comes to GDRDMA - These configurations support IBVerbs which feed into NCCL collective operations

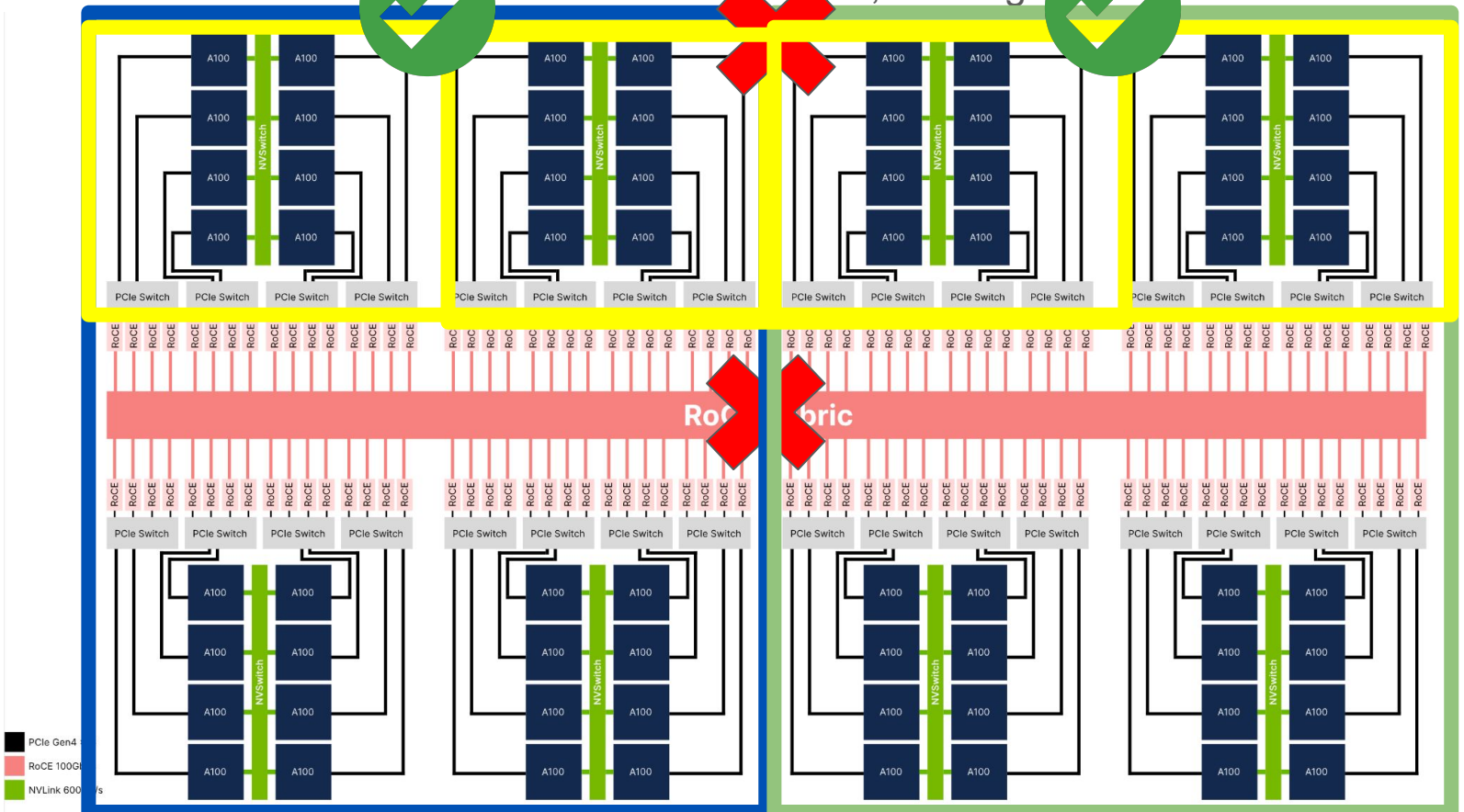
- AWS
  - EFA (Elastic Fabric Adapter), Libfabric
- OCI
  - RoCE (RDMA over Converged Ethernet), MLNX OFED
- Azure
  - IB (InfiniBand), MLNX OFED
- Coreweave
  - IB (InfiniBand), Fully Managed Kubernetes

Testing with NVIDIA nccl-tests or Meta param

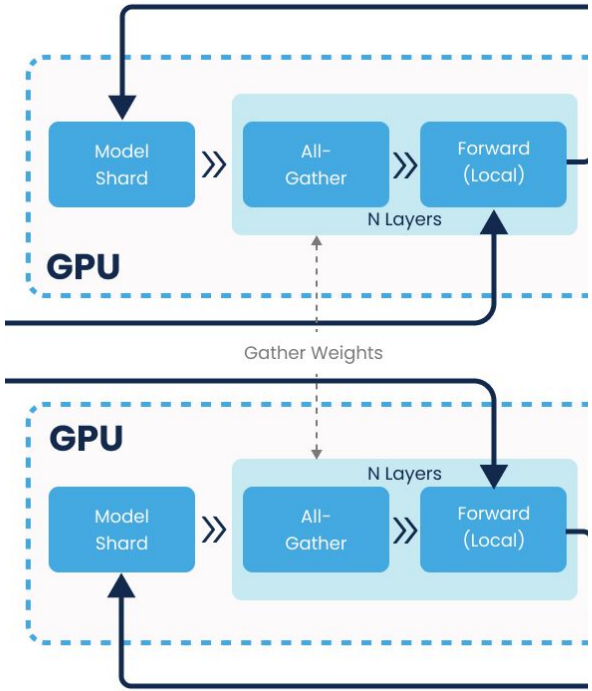


RDMA Network Switch Failure

Network switch failures caus[illegible]rtions within the cluster, causing [illegible] to fail with NCCL timeout



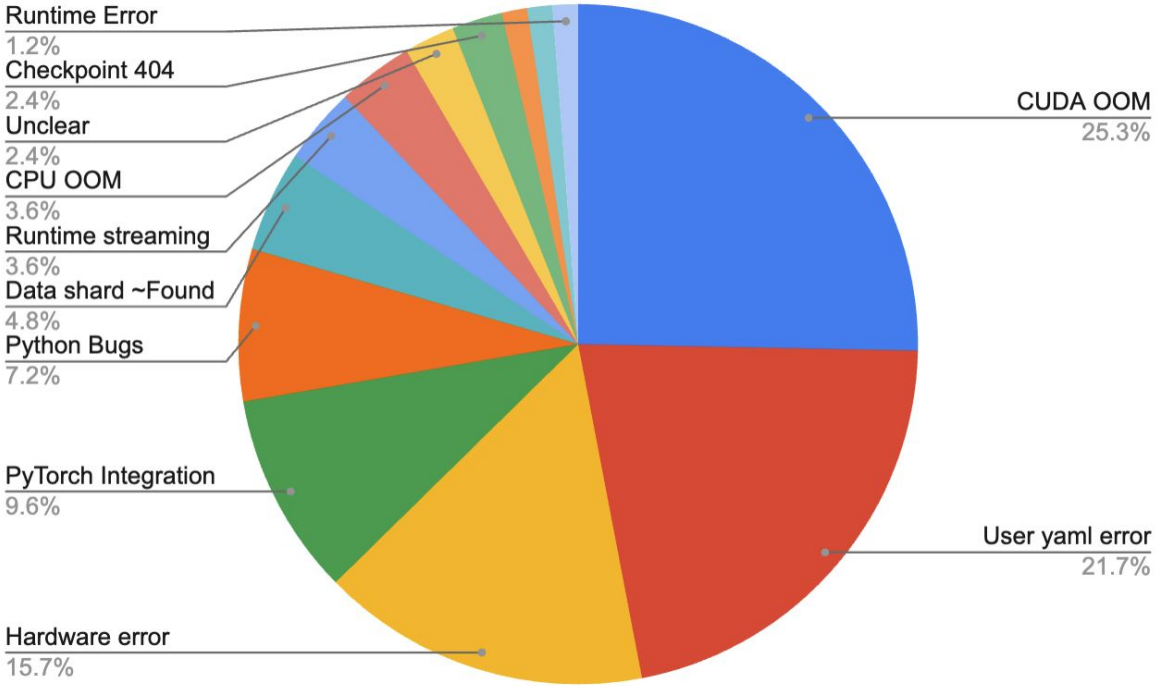
Complexity in NCCL Timeout Debugging




```
23 [rank22]:[E ProcessGroupNCCL.cpp:563] [Rank 22] Watchdog caught collective operation timeout: W
   NumelOut=545914880, Timeout(ms)=600000) ran for 600038 milliseconds before timing out.
24 [rank23]:[E ProcessGroupNCCL.cpp:563] [Rank 23] Watchdog caught collective operation timeout: W
   NumelOut=545914880, Timeout(ms)=600000) ran for 600025 milliseconds before timing out.
25 [rank24]:[E ProcessGroupNCCL.cpp:563] [Rank 24] Watchdog caught collective operation timeout: W
   NumelOut=545914880, Timeout(ms)=600000) ran for 600010 milliseconds before timing out.
26 [rank25]:[E ProcessGroupNCCL.cpp:563] [Rank 25] Watchdog caught collective operation timeout: W
   NumelOut=545914880, Timeout(ms)=600000) ran for 600032 milliseconds before timing out.
27 [rank26]:[E ProcessGroupNCCL.cpp:563] [Rank 26] Watchdog caught collective operation timeout: W
28 [rank27]:[E ProcessGroupNCCL.cpp:563] [Rank 27] Watchdog caught collective operation timeout:
   NumelOut=545914880, Timeout(ms)=600000) ran for 600081 milliseconds before timing out.
29 [rank29]:[E ProcessGroupNCCL.cpp:563] [Rank 29] Watchdog caught collective operation timeout:
   NumelOut=545914880, Timeout(ms)=600000) ran for 600078 milliseconds before timing out.
30 [rank30]:[E ProcessGroupNCCL.cpp:563] [Rank 30] Watchdog caught collective operation timeout:
   NumelOut=545914880, Timeout(ms)=600000) ran for 600004 milliseconds before timing out.
```

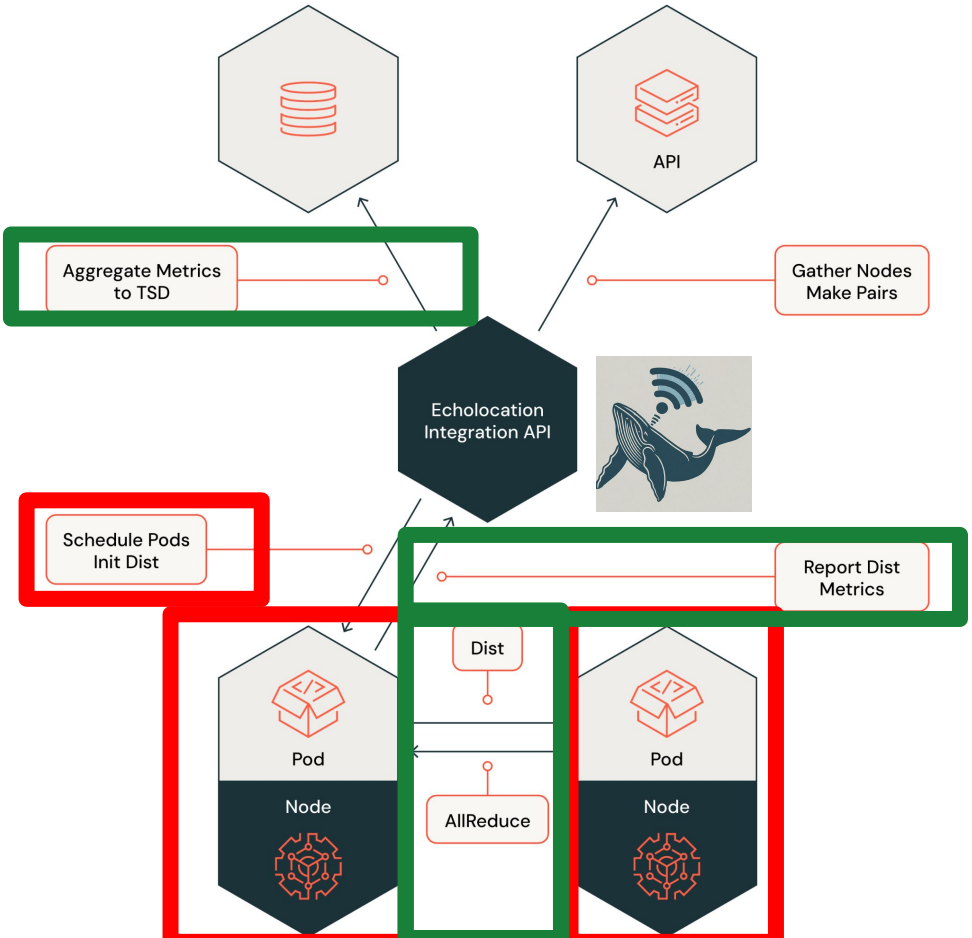
- NCCL timeouts may not be due to fabric issues
- Rank can timeout when collective operation not completed, due to issues w/ other rank.
- Key to debug is to look at stack trace from rank that did not reach the NCCL barrier

Most NCCL timeouts due to User Error



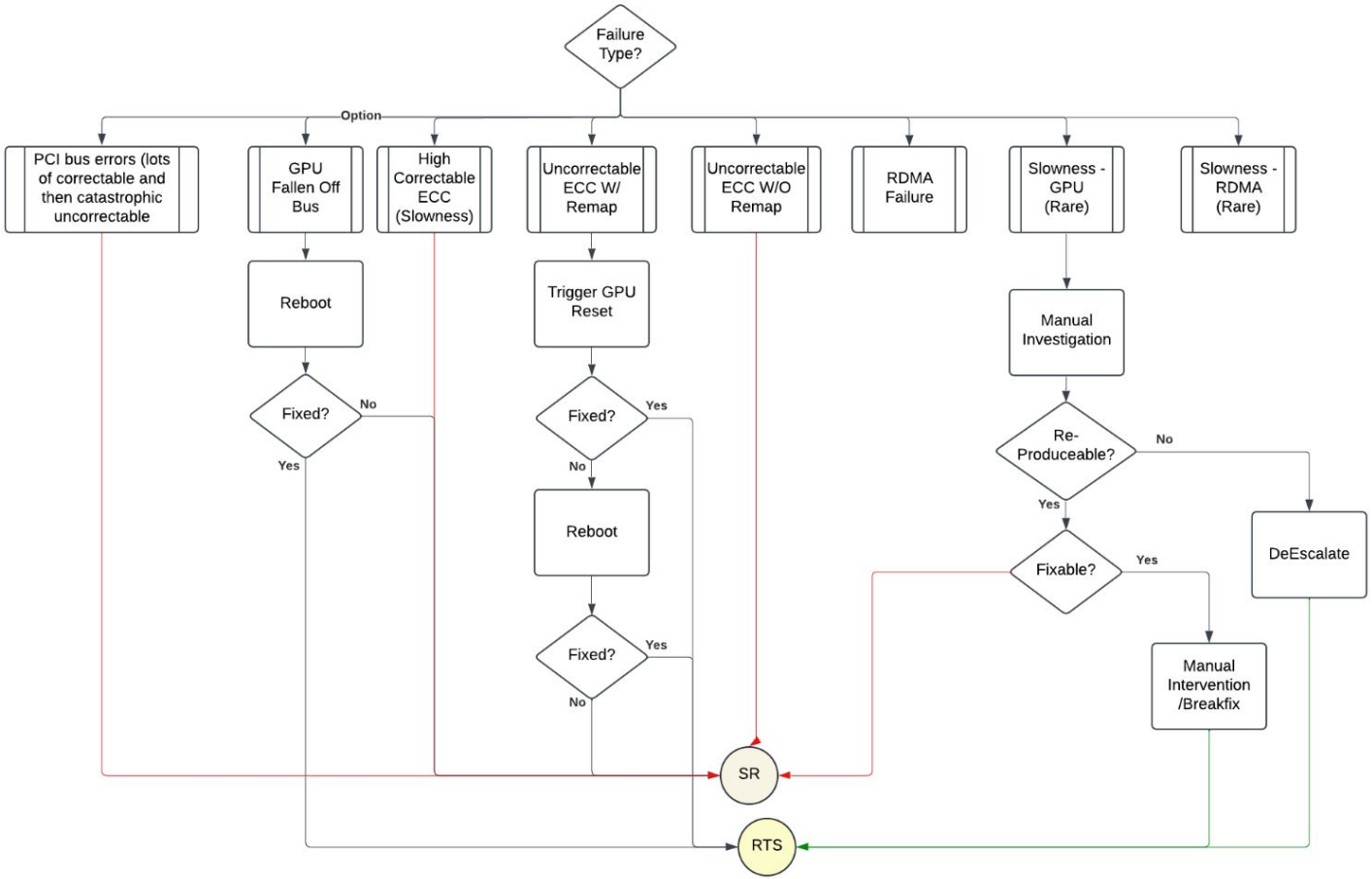
- Only 16% NCCL timeouts across all workloads (!DBRX) due to fabric issues
- Need for running periodic active checks to isolate real fabric issues

Active Fabric Check Design



1. Detection of fabric issues using two-round NCCL tests (w/ NVIDIA nccl-tests or Meta param)
2. Generic framework to add new health-check signals easily across (1x/2x/4x/8x) nodes

Summary


Failure Type?
Option
PCI bus errors (lots of correctable and then catastrophic uncorrectable
GPU Fallen Off Bus
High Correctable ECC (Slowness)
Uncorrectable ECC W/ Remap
Uncorrectable ECC W/O Remap
RDMA Failure
Slowness - GPU (Rare)
Slowness - RDMA (Rare)
Reboot
Fixed?
No
Yes
Trigger GPU Reset
Fixed?
Yes
No
Reboot
Fixed?
Yes
No
Manual Investigation
Re-Produceable?
No
Yes
Fixable?
Yes
DeEscalate
Manual Intervention /Breakfix
SR
RTS

Andrej Karpathy
@karpathy

...TLDR LLM training runs are significant stress-tests of an overall fault tolerance of a large computing system acting as a biological entity. And when you're shopping around for your compute, think about a lot more than just FLOPs and $. Think about the whole service from hardware to software across storage, networking, and compute. And think about whether the team maintaining it looks like The Avengers and whether you could become best friends.

9:10 AM · Mar 6, 2024 · **644.7K** Views

520 Reposts **63** Quotes **4,289** Likes **3,506** Bookmarks

databricks

We’re hiring!

https://www.databricks.com/company/careers/open-positions

Questions?

Please scan the QR code above to leave feedback on this session