

Distributed-training incident runbook

A printable operator workflow for multi-rank crashes, hangs, transport faults, and corrupted recovery state.

Incident ID _____ _____	Start time and timezone _____ _____	Severity _____ _____	Status _____ _____
Incident commander _____ _____	Evidence owner _____ _____	Job and allocation _____ _____	Affected service or users _____ _____
Framework and versions _____ _____	World size and topology _____ _____	Image or environment _____ _____	Checkpoint and data _____ _____

Impact statement

What is broken, who is affected, what state is at risk, and what must remain running

Declare and contain

Set one incident owner, stop evidence-destroying retries, and limit impact without widening the outage.

[]	Record the incident start time, workload ID, scheduler job ID, cluster, framework, world size, model, and checkpoint.
[]	Name an incident commander and an evidence owner. Keep one time-ordered decision log.
[]	Stop automatic retries after the first reproducible failure. Preserve the failed allocation until volatile evidence is captured.
[]	Drain only nodes or devices that can harm peer workloads. Record every containment command and its operator.
[]	State the user impact, affected jobs, checkpoint exposure, data integrity risk, and current recovery objective.

Evidence and initiating rank

Preserve evidence

Collect the first distinct event from every rank and every layer before reset, reboot, eviction, or log rotation.

[]	Save stdout and stderr separately for every rank with host, local rank, global rank, process ID, GPU UUID, and timestamps.
[]	Capture scheduler state, exit codes, node reasons, allocation details, container image digest, environment, and resolved configuration.
[]	Capture kernel, driver, Fabric Manager, NCCL, network, storage, and checkpoint logs for the same time window.
[]	Record nvidia-smi -q, topology, link counters, ECC state, DCGM diagnostics when safe, and nvidia-bug-report.sh.
[]	Hash copied logs and checkpoints. Keep the originals read-only and record clock skew between nodes.

Find the initiating rank

Separate the rank that failed first from ranks that only timed out while waiting.

[]	Normalize timestamps, then sort the first distinct warning or error from every rank and host.
[]	Identify the first rank that stopped making progress, exited, emitted a CUDA or kernel fault, or diverged from collective order.
[]	Treat collective timeout reporters as waiters until evidence shows one of them initiated the failure.
[]	Query NCCL RAS when available and compare repeated status snapshots for dead, missing, or non-advancing ranks.
[]	Write the initiating event, downstream symptoms, confidence, and competing hypotheses before changing the system.

Bounded diagnostic commands

journalctl -k --since '<start>' --until '<end>'
nvidia-smi -q && nvidia-smi topo -m
NCCL_DEBUG=INFO TORCH_CPP_LOG_LEVEL=INFO TORCH_DISTRIBUTED_DEBUG=DETAIL <bounded-control>
echo 'verbose status' nc localhost 28028
sha256sum <copied-log-or-checkpoint>

Triage matrix

Start with the literal symptom, then locate evidence that predates downstream timeouts and wrapper errors.

Symptom	Evidence that identifies the initiator	Controlled comparison
Collective hang or watchdog timeout	First non-advancing or missing rank, NCCL RAS snapshots, collective sequence mismatch, earlier rank-local fault.	Same workload with matched rank placement on a known-good node or fabric path.
One rank exits or segfaults	That rank's first CUDA, application, kernel, memory, or signal event before peer timeouts.	Same input and seed with the rank moved to known-good hardware, then with the workload held on the original device.
NCCL system or transport error	NCCL INFO or RAS output, socket or RDMA errno, interface selection, link counters, topology, and container limits.	Matched run on a verified interface or path while every application variable stays fixed.
CUDA OOM or allocator divergence	Per-rank reserved and allocated memory, input shape, batch, sequence length, fragmentation, and first allocation failure.	Same rank and device with one bounded memory variable changed and all inputs preserved.
Xid, ECC, or GPU disappearance	First kernel Xid, PCI address, GPU UUID, ECC or row-remap state, BMC events, power, thermals, and link history.	Vendor-approved device or slot isolation test after volatile evidence and peer workloads are protected.
Checkpoint read or write failure	First storage error, object version, size, checksum, writer rank, completion marker, filesystem or service response.	Read the same immutable artifact independently, then write a bounded artifact through a known-good path.

Initiating event and competing hypotheses

First causal event and receipt	Confidence and competing hypotheses

Controlled comparison worksheet

Run one controlled comparison

Change one variable while holding workload, seed, input, topology, versions, and observation window fixed.

[]	Write the hypothesis, independent variable, fixed conditions, expected result, stop condition, and evidence to retain.
[]	Prefer a known-good node, rank placement, fabric path, checkpoint, image, or dependency version as the comparison.
[]	Use diagnostic settings only for the bounded control. Record their overhead and remove them after the test.
[]	Repeat the exact baseline and control when the first result could be timing-sensitive.
[]	Accept ownership only when the failure consistently follows the changed variable or disappears under the matched control.

Hypothesis _____	Independent variable _____
Fixed conditions _____	Expected result _____
Baseline receipt _____	Control receipt _____
Result and repeat count _____	Ownership decision and confidence _____

Stop condition and rollback

Stop the control if safety, peer workloads, data integrity, or the declared time budget is threatened _____

Recovery, observation, and closure

Recover and observe

Restore service only after the causal condition changes and the recovery state passes integrity checks.

[]	Verify checkpoint size, checksum, readable metadata, expected step, and a bounded restore before resuming the full job.
[]	Canary the corrected configuration at small scale with the same production-shaped input and collective path.
[]	Record the exact remediation, versions, node set, checkpoint, start time, and rollback command.
[]	Observe the original signature, initiating warning, throughput, loss, GPU health, network errors, and checkpoint writes.
[]	Declare recovery only after the full observation window passes with no recurrence and expected progress.

Hand off and close

Leave a standalone receipt that another operator can verify without chat history or incident memory.

[]	Record the initiating evidence, causal claim, confidence, rejected hypotheses, controlled comparison, and result.
[]	Link immutable log, configuration, checkpoint, diagnostic, change, and monitoring receipts.
[]	Name follow-up owners, deadlines, rollback triggers, and the verification command for every open action.
[]	Record the observation window, recurrence count, restored capacity, affected users, and any remaining production risk.
[]	Schedule the postmortem and preserve the runbook copy with the incident record.

Official sources

[NVIDIA NCCL RAS troubleshooting](https://docs.nvidia.com/deeplearning/ncll/user-guide/docs/troubleshooting/ras.html) <https://docs.nvidia.com/deeplearning/ncll/user-guide/docs/troubleshooting/ras.html>

[NVIDIA NCCL troubleshooting](https://docs.nvidia.com/deeplearning/ncll/user-guide/docs/troubleshooting.html) <https://docs.nvidia.com/deeplearning/ncll/user-guide/docs/troubleshooting.html>

[PyTorch distributed debugging](https://docs.pytorch.org/docs/stable/distributed.html) <https://docs.pytorch.org/docs/stable/distributed.html>

[NVIDIA GPU Debug Guidelines](https://docs.nvidia.com/deploy/gpu-debug-guidelines/) <https://docs.nvidia.com/deploy/gpu-debug-guidelines/>

Use site safety procedures and vendor-specific recovery guidance. Diagnostic settings can add overhead and should be limited to a bounded control.