

# TorchPass AI Fault Tolerance

March 11, 2026 · [clockwork.io/blog/torchpass-workload-fault-tolerance/](https://clockwork.io/blog/torchpass-workload-fault-tolerance/)



At scale, distributed AI training is notoriously fragile. Just one link flap or GPU fault can crash an entire long-running job and leave hundreds or thousands of GPUs idle. With 1,024 GPUs, the cost of failures can add up to more than \$300,000 per month, not including the impact of lost revenue from delayed model releases.

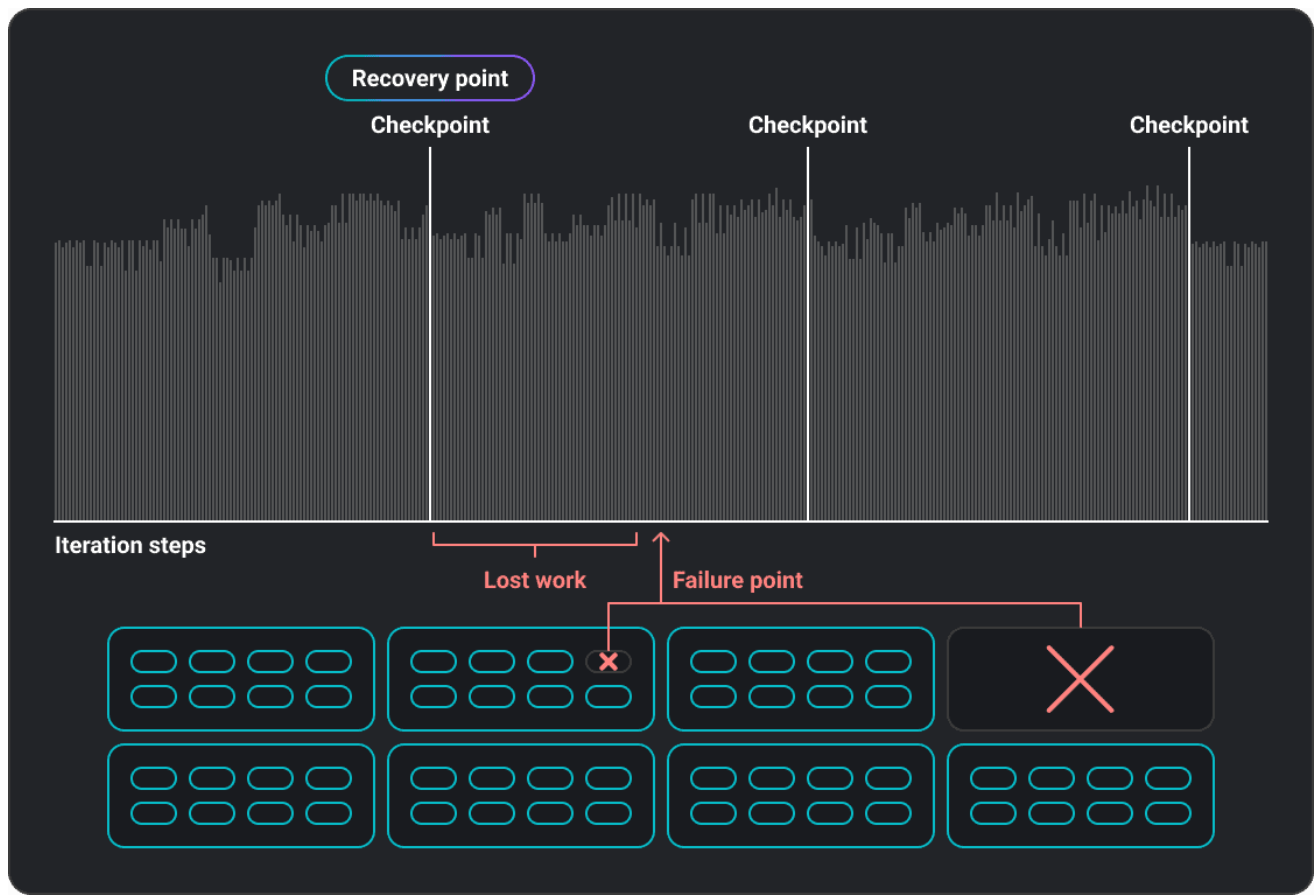
We built TorchPass – a system that delivers fault tolerance to AI training – to eliminate this waste entirely.

## AI Cluster Fragility

| Metric                                             | Source                               |
|----------------------------------------------------|--------------------------------------|
| 7.9 Hour Mean-Time-To-Failure (MTTF) for 1024 GPUs | <a href="#">Meta Fair HPCA 2025</a>  |
| 1.8 Hour MTTF for 16,384 GPUs                      | <a href="#">Meta Fair HPCA 2025</a>  |
| 419 failures over 54 days                          | <a href="#">Meta Llama 3 Report</a>  |
| 60% of jobs experience slowness                    | <a href="#">Alibaba SIGCOMM 2024</a> |

## Why the Cost of Failure is so High?

When a training run fails, the common recovery approach is to restart from the most recent checkpoint. But each time this happens, the computational work performed since the last checkpoint is lost entirely. For example, if checkpoints are taken every 3 hours, the average amount of lost work is 1.5 hours per incident.



Additionally, time is lost finding and provisioning a replacement node, loading the checkpoint and restarting the training loop. These factors are summarized in the chart below:

Factors Impacting the Cost of Checkpoint Restarts

| Factor                  | Description                                                                                                                             |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Lost computational work | The amount of computational work since the last checkpoint is lost and has to be recomputed (half the checkpoint frequency on average). |
| Node replacement time   | The time it takes to find and provision a replacement node (often compounded by having to apply patches, update firmware, etc.).        |

| Factor                         | Description                                                                             |
|--------------------------------|-----------------------------------------------------------------------------------------|
| <b>Checkpoint restore time</b> | The time it takes to restore the checkpoint from persistent storage.                    |
| <b>Job startup time</b>        | The time it takes to restart the training job across all nodes from the recovery point. |

As noted above, the cost of these failures is high. For example, in a 1,024 GPU configuration, the costs can add up to \$307,000 per month (assumes \$3/GPU/hour, 1.5 hour CP frequency, 10 mins node replacement, 5 mins checkpoint restore, 5 mins job startup and 7.9 hour MTTF).

## The TorchPass Approach to Training Fault Tolerance

The overarching goal of TorchPass is to allow training to continue, with no loss of training progress, through any kind of disruption scenario (more on this later).

TorchPass is entirely software-based, runs anywhere (cloud and on-prem) and supports popular training frameworks and schedulers including TorchTitan, Megatron-LM, DeepSpeed, Slurm and Kubernetes.

TorchPass is made up of two components: network fault tolerance and live GPU migration. These are described below.

## Network Fault Tolerance via Path Failover

We released the first phase of fault tolerance in September 2025 to address link flaps and other network issues – which represent some of the most common causes of failure (see: Llama 405B training details). This technology is already in use by Hyperscalers, Fortune 1000 enterprises and leading Neoclouds who are deploying our solution in clusters with over 100,000 GPUs.

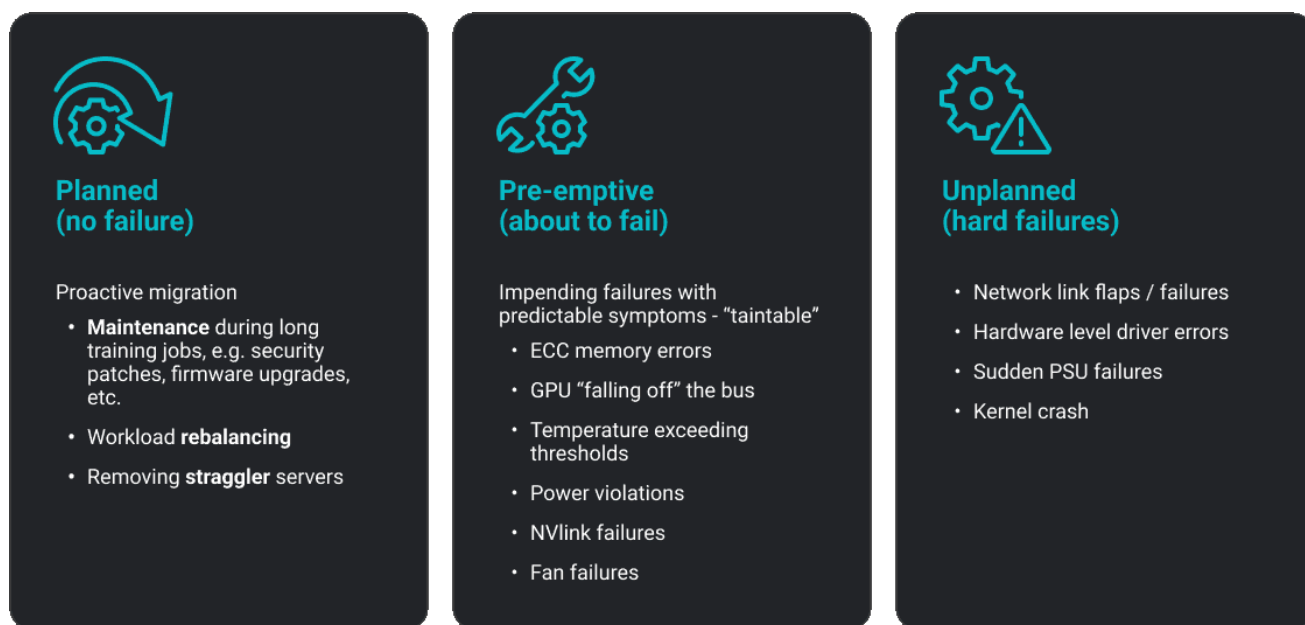
Network fault tolerance is implemented as a NCCL (NVIDIA Collective Communications Library) or RCCL (AMD's ROCm Collective Communications Library) net plugin that transparently intercepts failures and reroutes traffic onto alternative paths, allowing training to continue. There can be a small performance drop due to fewer available network paths, however as soon as the issue clears, full performance is restored. If the issue persists, (spoiler alert!) TorchPass can transparently migrate impacted rank(s) to spare GPU(s) to ensure full network bandwidth.

## Introducing TorchPass Workload Fault Tolerance via Live GPU Migration

Today we announced the next phase of TorchPass AI Fault Tolerance to provide resilience for a broad class of disruptions including GPUs falling off the bus, GPU memory issues, software/driver bugs and even full node failures. When one or more components fail, TorchPass performs a live GPU migration of impacted ranks to resources drawn from floating or dedicated spares. The following animation illustrates this at a high-level and further details are provided below in the section "How TorchPass Works"

### Migration Use Cases

TorchPass supports three use cases for migration:



- **Planned** (no failure) – for situations where proactive migration is preferred over taking resources offline. E.g. security patching, firmware updates, preventative maintenance, workload rebalancing and removing straggler GPUs or servers.
- **Pre-emptive** (about to fail) – for issues that have predictable symptoms that occur before failure. E.g. ECC memory error rates or temperature increasing above a threshold. This class of problems involves "tainting" the impacted resource(s) to indicate imminent failure.
- **Unplanned** (hard failures) – for failures that occur suddenly and without warning. E.g. kernel crashes, complete power failures, driver errors, etc.

## Dedicated or Floating Spares

TorchPass seamlessly manages spare resources to ensure rapid recovery from failures.

Customers may choose to designate dedicated spare GPUs, guaranteeing immediate failover capacity. However, many teams prefer not to leave expensive GPU resources idle. To address this, TorchPass also supports floating spares, dynamically drawing replacement capacity from any

available resources in the cluster or pulling them from lower priority jobs. This maximizes overall utilization while preserving fault tolerance.

In addition, since many disruption conditions are resolved through restart or reboot, TorchPass can even migrate workloads back to their original components once they return to a healthy state.

## Combining Network Fault Tolerance with Live GPU Migration

The two components of Clockwork's AI fault tolerance solution – network fault tolerance via path failover and live GPU migration – are designed to work together.

When a network disruption such as a link flap occurs, the first line of defense is immediate path failover, allowing the job to continue running without interruption. However, as described above, there will be a small drop in performance due to fewer available network paths. If the fault resolves itself, full performance is automatically restored by falling back to the original network path. If the fault persists, a live GPU migration can be performed to resources with full network connectivity. In this way, a hard network failure is immediately mitigated and then converted into a graceful pre-emptive live GPU migration.

## Failure Detection

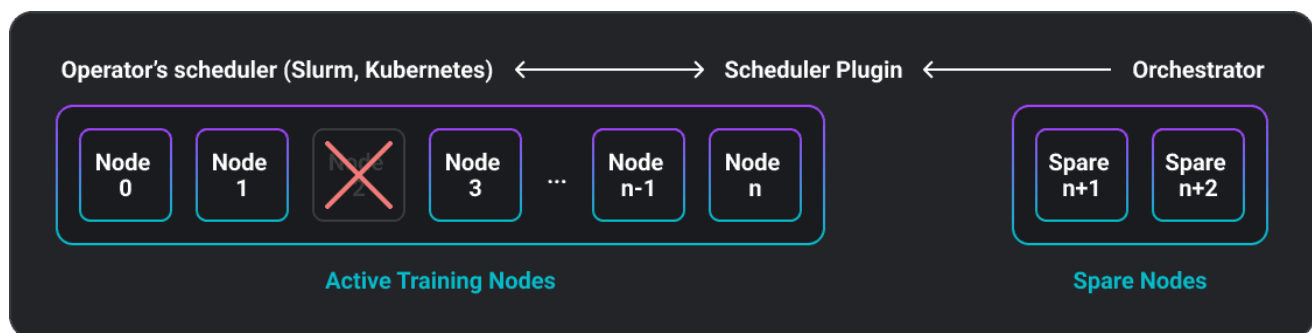
The process of identifying a problem (hard failure or pre-emptive disruption) is independent of the TorchPass implementation. This means TorchPass supports Clockwork mechanisms to catch problems (our AI observability solution is constantly being improved to cover a comprehensive set of such issues) or mechanisms that are already being used by our customers (this is common with many of our Neocloud and large enterprise customers).

## How TorchPass Works

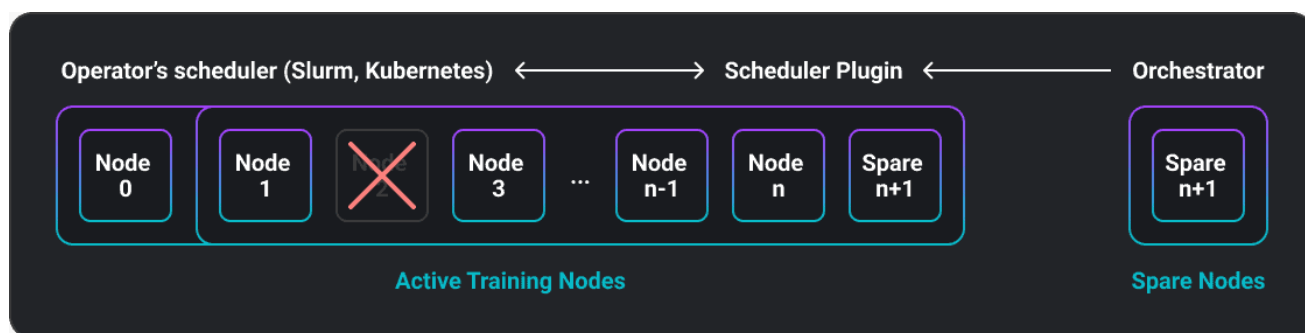
TorchPass is made up of two primary components:

- **TorchPass Orchestrator** – the central brain that communicates with the TorchPass scheduler plugin to track jobs and spare resources. It manages the migration process when a failure (planned or unplanned) occurs.
- **TorchPass Scheduler Plugin** – the adapter between the TorchPass Orchestrator and the cluster’s native scheduler (e.g. Kubernetes or Slurm) API.

The following explains what happens when a migration occurs and assumes there is a cluster with  $n$  active training nodes and two spare nodes. For this example, we’ll illustrate the complete failure of node 2 (note that the process for a GPU failure is similar, except that only the impacted worker is migrated rather than the full node):



The Orchestrator tracks spares and the jobs in progress via the scheduler plugin. Upon failure (or tainting), the orchestrator schedules the migration of the failed node to a spare node. Work is paused briefly while the migration takes place (with full state transfer). Once complete, the Scheduler plugin updates the main scheduler’s job definition to reflect the swap, and training progress is resumed. At the end of the process, the system looks like this:



## Going a Little Deeper Under the Covers

As described above, TorchPass supports three migration use cases: planned (proactive migrations), pre-emptive (imminent failures) and unplanned (sudden failures). From an implementation perspective, pre-emptive and planned migrations look the same because a failure has not actually occurred, and so the migration can be performed in a completely controlled manner by notifying TorchPass via “tainting” the resource(s). For simplification, we'll call them both “planned migrations”.

### Planned Migration

TorchPass supports two mechanisms for planned migrations:

- **Model Aware** – this requires importing a TorchPass library into the training code and registering existing checkpoint and restore functions with the TorchPass library (a simple process involving just a few lines of code). The benefit of this is that it provides the fastest type migration (e.g. 30 seconds).
- **Model Transparent Planned Migration** – this does not require any code or model changes but takes a little longer (e.g. 2 minutes) than the model-aware variant.

The process for both is similar: all workers agree on a migration boundary, training is briefly quiesced, state is copied from the leaving worker(s) to the joining worker(s), and training can then

transparently resume. The difference between the two involves how state is captured: with model aware migration standard checkpoints can be taken, whereas with model transparent a combination of system (for CPU) and CUDA (for GPU) snapshots are used.

## Unplanned Migration

The unplanned migration flow is similar to that of planned, but with an important difference: state cannot be captured from workers or nodes that have already failed. Instead, TorchPass relies on the fact that most training environments utilize some form of data parallel replicas (e.g. DDP, FSDP, HSDP, TP, etc.). So instead of capturing state using model aware checkpoints or system-level snapshots, state for the failed workers is reconstructed from just-in-time checkpoints taken from the healthy workers.

## Why TorchPass Matters

### Ship Models Faster

Network problems, GPU faults, software/driver bugs and even entire node failures will no longer crash training jobs. This removes wasted work and lost GPU cycles from impacting the time it takes to complete model training.

Using the example from earlier in the blog, without TorchPass approximately 3 hours of training progress would be lost per day in a 1024 GPU cluster (assumes 1.5 hour CP frequency, 15 minutes bring-up time and 7.9 hour MTTF). Using Clockwork TorchPass with model transparent planned migrations (the slower of the two planned migration options), this would typically drop to less than 10 minutes per day – or 95% less time wasted. In addition, proactive maintenance (security patching, software updates, etc.) will no longer take away from model training time.

## Increase GPU ROI

Continuing the same example, the cost of lost GPU time in a 1,024-GPU environment drops by approximately \$300,000 per month when using TorchPass!

## Accelerate the Transition to Nvidia GB200, GB300 and Vera Rubin systems

Many of our customers are in the process of evaluating or implementing rackscale GB200 and GB300 NVL72 systems and preparing for next generation Vera Rubin NVL144 systems. These configurations deliver industry-leading energy efficiency and performance per GPU dollar spent. However, their dense, tightly coupled architectures amplify the impact of small failures on distributed training workloads.

Since TorchPass now makes reliability a software-defined property, it enables the safe deployment and maximized ROI of these new systems without compromising training reliability.

## Runs Anywhere

TorchPass is 100% software-based, runs anywhere and supports popular training frameworks and schedulers including TorchTitan, Megatron-LM, DeepSpeed, Slurm and Kubernetes.

## Alternative Approaches to Fault Tolerance

### Checkpoint Recovery

As described above, the most common method to protect against training failure is to take frequent distributed checkpoints during training and recover from the most recent checkpoint in the event of a failure. Although effective, this approach has a key limitation: upon failure, all computational work performed since the last checkpoint is lost and must be recomputed from that checkpoint.

Additionally, recovery introduces operational delays, including provisioning a replacement node, reloading the checkpoint, and reinitializing the distributed training process.

## TorchFT

In late 2024, TorchFT was announced as a solution providing fault tolerance at per-step granularity. It requires a DDP (distributed data parallel) configuration and treats each training step as a distributed transaction. Upon failure, an entire replica group is removed and training continues without processing samples assigned to that group until it has been replaced. There is also a significant per-iteration overhead during normal training due to TorchFT's use of the Gloo Collective Communication Library for cross-replica all-reduce operations.

A detailed comparison between TorchPass and TorchFT can be found here, along with testing results comparing the approaches here.

## Conclusion

As models and clusters continue to scale and new generations of GPU hardware enter mainstream deployment, failure becomes more than just prohibitively expensive, it becomes a barrier to delivering advanced models on time.

Implementing fault tolerance, in a seamless and flexible manner, will become even more of a necessity. This is why we are so excited to offer TorchPass AI fault tolerance to our customers today!