

TorchPass Snapshots: Platform-Controlled Snapshots for Distributed Training

Preserve the state of a running training job without touching the training code.

When running a GPU fleet, platform teams often need to reclaim GPU nodes for higher-priority work, consolidate fragmented allocations or vacate machines for maintenance. Similarly, cloud providers routinely reclaim spot instances. These are routine events: On average, Meta's training scheduler preempts each workload once which leads to about 100,000 scheduling attempts daily [12]. The usual scheduling controls available to a platform team are to suspend or requeue a job through Slurm or terminate Pods using Kubernetes. These controls can free the GPUs but they do not, by themselves, let the platform save the running job's state. Preserving progress and resuming correctly, then, depends on the application implementing its own response to a platform signal in the form of automated application checkpoints [4–7].

Cloud operators and platform teams should instead be able to release GPUs without discarding the running job, by saving the job's state as a platform capability - not just an application capability. Safe interruption requires the workload to coordinate its workers, save complete state before the machines disappear, and reload that state correctly afterward. Application checkpoints can be automated to achieve this but where that integration is absent or unverified, the platform must either defer reclamation or accept the risk of losing work. This is particularly true for Cloud operators whose tenants need to own the automated application response. **The operational gap is a dependable way to preserve an eligible running job without making that capability depend on application-specific checkpoint callbacks [3].**

TorchPass snapshots give platform teams that control. It is in preview with planned availability in Q4 2026. It coordinates a speed-optimized consistent snapshot of a distributed training job's supported process and GPU state without changing the training code. Once the complete image is protected, training can resume or the platform can reclaim nodes for higher-priority work, consolidation or maintenance. The job can later resume unaffected on available capacity.

[TorchPass](#), Clockwork's live GPU migration product, uses the same capture-and-restore mechanism to move workers onto an available spare. TorchPass snapshots separate capture from restore so replacement GPUs do not have to be ready when the originally allocated nodes are released. Snapshots can also be taken periodically with minimal impact, allowing warm recovery after an unexpected interruption, complementing application checkpoints as an insurance policy.

1. Operators and Platform teams must preserve jobs when releasing nodes

Large training jobs are interrupted for four kinds of reasons: Schedulers preempt lower-priority jobs to make room for higher-priority work, operators move jobs to fix placement or performance problems, operators drain hosts for repair or maintenance and failures occur without notice. Meta's Llama 3 pre-training run recorded 466 interruptions in 54 days on 16,384 GPUs. Of these, 419 were unexpected, and about 78 percent of those were attributed to confirmed or suspected hardware issues [1].

1. **Reclaim capacity.** Preemption interrupts lower-priority work so another workload can use its resources [6, 7, 12]. The operator may need to serve an urgent job or return borrowed capacity to its owner [6, 7]. A time-limited reservation creates a similar deadline. Amazon SageMaker training plans, for example, begin terminating running instances when 30 minutes remain in the reservation [20]. Preemptible cloud capacity gives less warning. Amazon Elastic Compute Cloud (EC2) Spot Instances get a best-effort two-minute notice [8], Google Cloud Spot virtual machines (VMs) up to 30 seconds [9] and Together AI's preemptible compute up to five minutes [10]. The running job needs a recovery point before its allocation ends.
2. **Optimize placement.** Free GPUs may be spread across racks while a waiting job needs machines in one network block. An operator may need to move smaller jobs to gather that capacity [7, 21]. After a failure, a job may need a replacement in its assigned rack when none is free [21]. Moving a healthy job out of that rack can make room for such a replacement, only if the healthy job is preserved during the move.
3. **Preserve eligible jobs during the drain window.** An operator may decide that a node needs repair, a reset, a reboot or a driver upgrade. The operator stops new work from starting on the node. Running jobs then finish, or the operator removes them with a short grace period [4, 5, 11]. Google Cloud performs host maintenance about every two weeks, and compute instances with attached GPUs "can't be live migrated" [22]. The operator needs to preserve eligible jobs within that window, and needs to know when a job cannot be preserved in time. Operators and platform teams send a signal (Kubernetes taints; Slurm Drain state), hoping that the application teams have automated mechanisms that save training state before the process exits. Where that response is missing or unverified, the training job exits without a fresh recovery point.
4. **Periodic platform snapshots as a complement to application checkpoints.** For sudden unplanned failures, recovery typically depends on periodic application-checkpoints as the basis for recovery ([TorchPass](#) live GPU migration facilitates non-disruptive training even in the event of sudden failures). Periodic infrastructure snapshots provide an additional recovery option for eligible jobs. The operator chooses a frequency that balances the protection gained against the interruption and copying costs.

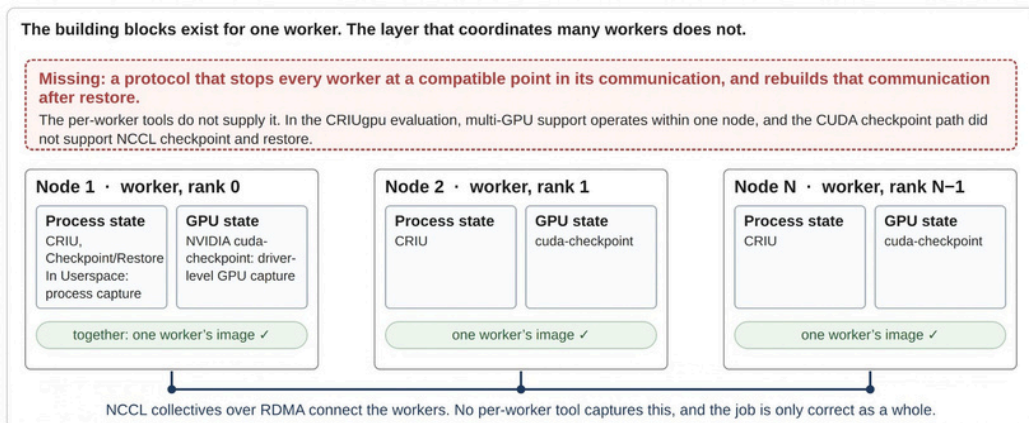
These situations require a dependable way to capture the running state of supported jobs without model-specific "save" code. Before releasing a machine, the operator needs to know that the complete job can resume from the preserved state. Event-time capture must finish the required preservation work before the machines are removed. Periodic capture must impose an acceptable training pause. Saved images must survive the intended failure and remain usable on compatible capacity. For a job spread across many nodes, that requires coordination among all of its workers.

2. The challenge: A single GPU or node's state is not a training snapshot

A distributed training job contains workers that depend on each other's progress. Pausing one worker can prevent its peers from completing their communication. The platform must bring the workers to a consistent stopping point before capture. It must also preserve the information needed to reconnect them after restore. NVIDIA's CUDA checkpoint mechanism and CRIU provide the building blocks for saving supported GPU and process state. Coordinating those captures across a running multi-node job requires an additional layer.

2.1 The missing layer is distributed coordination

NVIDIA's cuda-checkpoint utility supplies driver-level capture and restore of supported GPU state [23]. CRIU, Checkpoint/Restore In Userspace, supplies the corresponding Linux process checkpoint. Together, they provide the foundation for preserving a



GPU worker. They do not, by themselves, supply the protocol that stops a distributed training job in a consistent state. The CRIUgpu paper identifies this distinction explicitly [14]. CRIUgpu’s multi-GPU support operates within one node, and the paper states that multi-node workloads require additional synchronization across machines. When the paper was written, cuda-checkpoint did not support checkpoint and restore of the NVIDIA Collective Communications Library (NCCL), which carries the communication between training workers.

The problem appears before any memory is copied. Suppose one worker stops after completing a collective operation, while another has already entered the next collective. The second worker can now wait indefinitely for the worker that has been frozen. Sending a pause command to every server is therefore not enough. The workers must reach a compatible stopping point in their communication, rather than merely stop at approximately the same time. The unit of correctness is the communicating job, not the individual process.

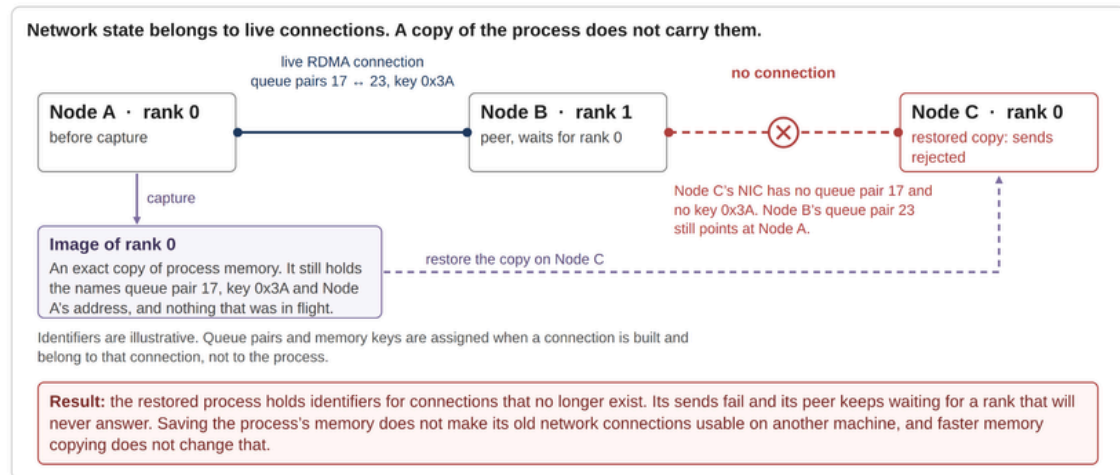
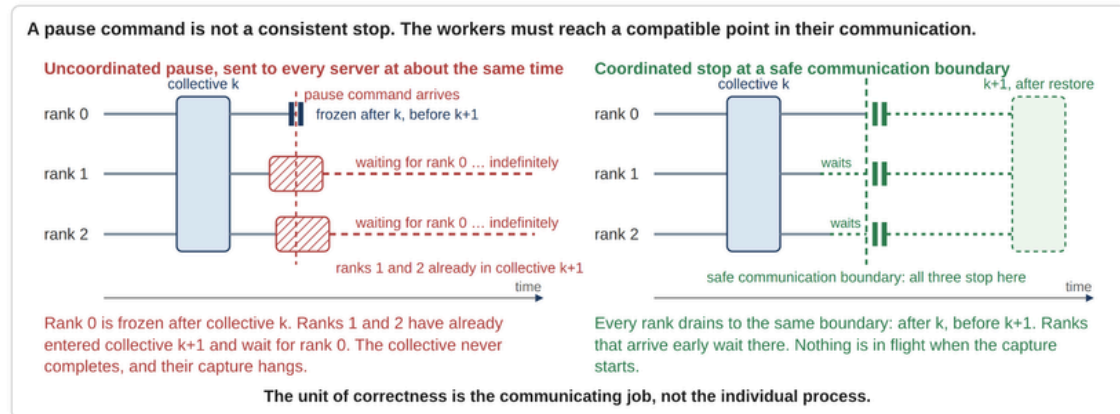
The network adds a second constraint. Remote direct memory access, or RDMA, uses communication endpoints called queue pairs and access keys for registered memory. The network adapter creates these resources for the running processes. Their identifiers can change when the resources are recreated. Saving the process’s memory does not make its old network connections usable on another machine. The snapshot mechanism must complete outstanding communication, remove transport resources that cannot survive capture, preserve the information needed to reconstruct communication, and rebuild

it around the restored workers. Faster memory copying cannot substitute for this coordination.

2.2 Warm-starting an inference worker does not establish live training recovery

Inference snapshots demonstrate why preserving an initialized runtime is valuable. NVIDIA Dynamo Snapshot captures an initialized GPU worker so that subsequent workers can start without repeating the full initialization sequence. Its version 1.4.2 documentation marks single-GPU vLLM and SGLang as supported, labels vLLM across several GPUs on one node “Highly experimental” and labels multinode configurations “Work in progress” [16]. NVIDIA notes that these checkpoints provide support for capturing the startup of an LLM inference engine and reports no training results - a platform that wants to release a running training job still needs an orchestration layer that coordinates across worker nodes.

That distinction is about the state being preserved, not simply the number of GPUs. Reusing an inference engine’s initialized state avoids repeated startup work. Preserving a running training job must also retain the workers’ evolving execution state at a mutually consistent point. A fast restore of an initialized worker does not demonstrate that an active distributed computation can be stopped, released and resumed correctly.



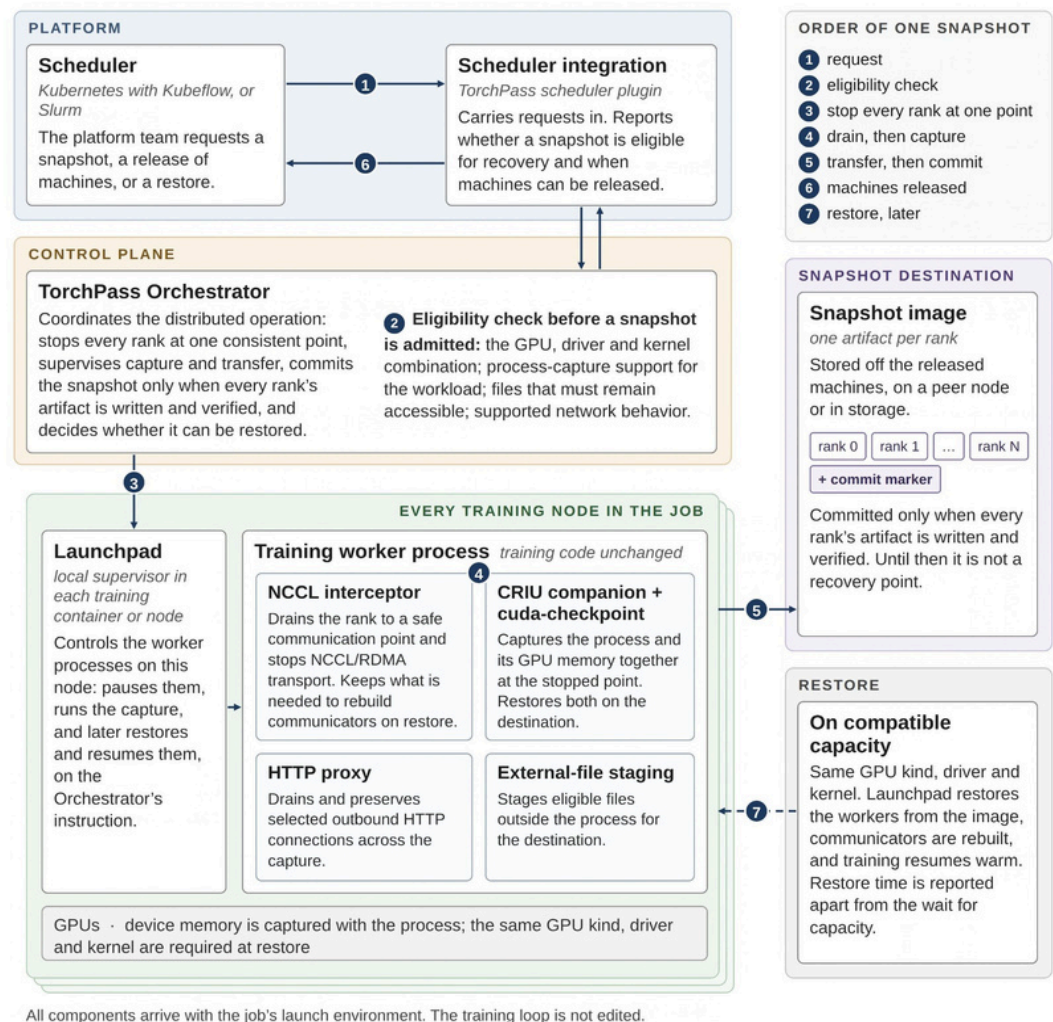
2.3 Training snapshots must obey constraints beyond distributed consistency

Distributed consistency is necessary, but it is not the whole product. The pause must be short enough for the operator's intended use, and the saved job must remain eligible for restore. A fast memory copy does not establish how long the job stops, when its machines can be reused, or how long it takes to resume useful training. The underlying tools also impose conditions that the platform must enforce - including supported memory types, GPU, driver and kernel compatibility constraints. A platform service must check those conditions before capture and restore, preserve the complete snapshot outside the machines being released, and distinguish a valid recovery image from a partial or failed capture. Creating snapshot files is not the same as making a job safe to release.

These are the gaps TorchPass snapshots address. TorchPass snapshots make the coordinated preservation of a distributed training job an operation the platform can control. The platform can then manage the lifetime and placement of the running job without requiring changes to the training code.

3. How TorchPass snapshots work: preserve the running job, release its machines

TorchPass snapshots give the platform a way to preserve a running distributed job before releasing its machines. It uses TorchPass's coordination and capture components, installed through the job's launch environment. TorchPass snapshots work in two modes. Infrastructure snapshots, described here, need no changes to the training code. Rolling application checkpoints, described in Sections 4.4 and 5, use the framework's own checkpoint integration. The operator can request a snapshot for a drain or schedule periodic snapshots for guaranteed recovery protection. The training code needs no model-specific checkpoint callbacks.



3.1 Capture the workers at a consistent point

Before capture, TorchPass snapshots check that the job and its environment support the operation. This includes the GPU and software combination, process-capture support, network behavior and required files. The orchestrator coordinates the operation through local supervisors on the training nodes.

The NCCL interceptor brings the workers to a consistent point in their communication. Outstanding transfers finish before capture. TorchPass snapshots record the communicator identities and sequence information needed to reconnect the workers later. Live NCCL and RDMA transport state is not carried in the image. TorchPass snapshots release it before capture and rebuild the communicators from the recorded information, whether the workers resume on the same machines or are restored elsewhere. Every required worker participates in this coordinated pause.

TorchPass snapshots also prepare the worker's supported connections to external services. It drains and suspends selected outbound HTTP traffic before process capture. After restore, the network-facing helpers are updated for the destination and supported HTTP activity resumes. The remote service accessed over HTTP itself remains outside the snapshot.

Required files are checked separately. TorchPass snapshots identify files that will be missing at the destination and stages eligible files for transfer with the process images. Other required files must remain accessible in the restored environment. Files whose contents affect the resumed job must also keep the versions it expects. Once communication and these dependencies are prepared, CRIU captures the Linux processes and their memory. NVIDIA's CUDA checkpoint mechanism captures supported GPU state. The image retains the initialized runtime, including the model and optimizer state held within it. Capture can occur partway through a training step. The saved workers represent one consistent point in the distributed job.

3.2 CRIU/CUDA checkpoint optimizations keep periodic snapshot pauses short

For large-model jobs, a TorchPass snapshot pauses training for only ~20 seconds. This has been achieved through a set of optimizations that reduce the overhead of the standard CRIU and CUDA checkpoint path. It moves bulk GPU memory through a significantly faster copy path and prepares reusable resources before training stops. Capture work runs in parallel across GPUs. Independent preparation and copying stages also overlap. These changes shorten the pause while preserving the supported process and GPU state.

Once a stable image exists, TorchPass snapshots repopulate GPU memory and resume the existing processes on the same machines. Replication and durable persistence continue in the background. The pause covers coordinated capture and local resume. Off-node protection, waiting for capacity and restoring elsewhere require separate time. The operator chooses how often to take periodic snapshots. At thirty-minute intervals, the pauses consume about ~1.1% of wall-clock time. At ten-minute intervals, they consume about ~3.3% of wall-clock time. The right frequency depends on the job's existing protection and how much work a failure could force it to repeat. For an operator, this means a small performance tax can be imposed in return for a guaranteed safe fallback, without a tenant having to modify their training code.

3.3 Keep jobs running when releasing its machines

With compatible replacement capacity ready, TorchPass platform migration moves the departing workers and rebuilds communication. The whole job stays paused until the moved workers resume. The departing node can then be released. For deferred restore, TorchPass snapshots save a complete distributed snapshot to storage outside the machines being released. Copies must survive removal of the machines they protect.

3.4 Restore the job on compatible capacity

Before restore, TorchPass checks the saved image against the destination. Restore keeps the captured GPU count [27] and model layout. The GPU type, driver, kernel, container environment and required files must meet the snapshot's compatibility requirements.

TorchPass snapshots restore process and GPU memory, reconnect the workers and resume the captured runtime. After a planned release, the job continues from the point at which it stopped.

4. Platform snapshots and model checkpoints: complementary protection, shared control

TorchPass snapshots complement application checkpoints rather than replacing them. Checkpoints provide the portable recovery foundation; they keep training recoverable when the environment changes. Snapshots preserve the running job so that many routine infrastructure events, such as repairs, reboots, reclaims and consolidation, need not trigger a cold restart. **Together, they provide warm recovery when compatibility holds and a fallback when it does not.**

4.1 Checkpoints preserve training progress; snapshots preserve execution

An application checkpoint records logical training state: model weights, optimizer state, training progress, random-number state and data-loader position. The application or framework defines what must be saved. Recovery reconstructs the model and runtime in a fresh process and loads that state. Its completeness depends on the application capturing everything required to continue correctly.

A TorchPass snapshot preserves the supported process and GPU runtime, including the training state inside it, without model-specific checkpoint callbacks. Recovery restores memory and rebuilds communication rather than repeating the full initialization sequence. The trade-off is destination compatibility and retention of the captured job topology. A model checkpoint supports reconstructing the job; a platform snapshot supports continuing the captured job.

4.2 A fast checkpoint save does not eliminate a cold restart

Asynchronous checkpointing reduces the save-side interruption by creating a stable copy of training state and persisting it while training continues [29]. It does not preserve the initialized process, so recovery still starts a fresh runtime, and that restart has its own cost. AWS reports that conventional checkpoint recovery took 4 minutes 52 seconds at 256 GPUs and 15 to 30 minutes at 2,304 GPUs [3]. TorchPass snapshots address that separate recovery cost: its value is warm restore and platform control.

4.3 Checkpoints cross boundaries that snapshots cannot

Application checkpoints can support different hardware or software environments that run the application and framework. Where the checkpoint format supports it, they can also redistribute saved state across a different GPU count or parallelism layout [29]. A runtime snapshot preserves its captured configuration rather than converting it into a new one.

Driver maintenance illustrates the distinction. Upgrading a GPU driver means unloading its kernel modules, so every GPU workload must leave the node first [11]. A snapshot lets the platform vacate machines and resume the job on compatible capacity. Moving onto an incompatible driver or kernel build still requires an application checkpoint and a fresh runtime.

Durability remains separate from portability. Either recovery image can disappear with its only copy. Peer placement and durable storage determine which failures it survives.

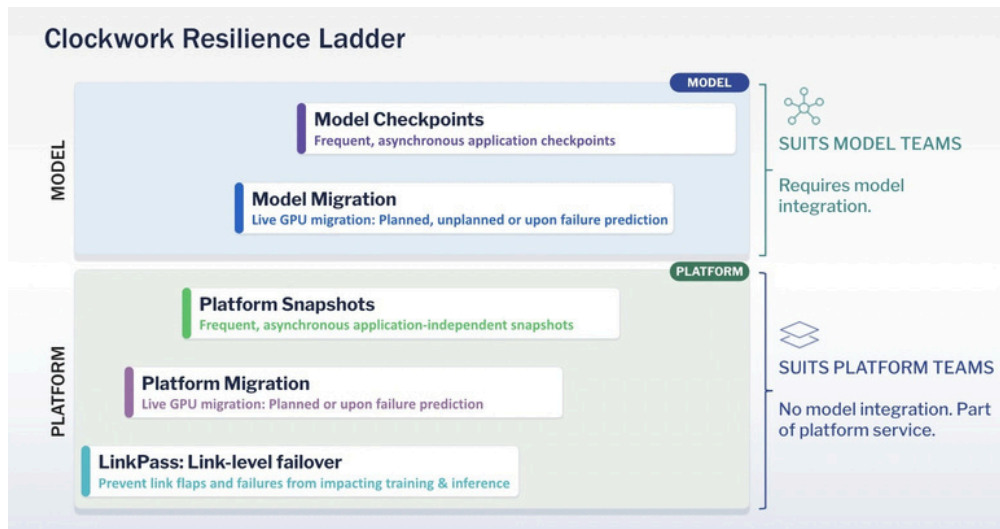
4.4 One orchestration layer, distinct responsibilities

Clockwork manages both mechanisms through TorchPass. The snapshot path requires no training-code changes. The application-checkpoint path retains the framework integration that declares state and establishes a consistent capture point. The training team remains responsible for checkpoint completeness; the platform manages triggers, background persistence, replication and recovery eligibility.

Recovery selection considers freshness, surviving copies, destination compatibility and expected recovery time, not simply the newest timestamp. A protected, compatible snapshot provides the warm path; a committed application checkpoint provides the fallback.

5. The resilience ladder

Resilience should not depend on one recovery mechanism or on every training team implementing the same interruption workflow. Clockwork's resilience ladder combines protection at two levels: infrastructure mechanisms that platform teams can deploy without changing training code, and application-aware mechanisms that use framework integration to provide additional recovery options.



5.1 One orchestration layer; complementary approaches to resilience

Platform-controlled solutions give platform teams the ability to preserve eligible workloads through infrastructure without requiring each model team to build a custom response:

- **LinkPass keeps a network problem from becoming a job restart.** It redirects affected communication onto healthy paths so the job can continue operating through covered link flaps and failures.
- **TorchPass platform migration moves execution before the source is lost.** For planned maintenance or predicted degradation, it transfers process and GPU state onto compatible replacement capacity without application checkpoint callbacks. The source must remain healthy enough to participate, and replacement capacity must be available [28].
- **TorchPass rolling infrastructure snapshots separate preservation from immediate replacement.** The platform can capture a job for planned suspension, release its machines once the image is protected, and restore it when compatible capacity returns. Previously committed snapshots also provide a warm recovery option after an abrupt failure, with lost work determined by the age of the snapshot.

Application-aware protection uses model integration:

- **TorchPass model migration extends recovery to supported unplanned failures.** Application integration identifies the training state needed to continue. TorchPass can then coordinate recovery on replacement capacity, even after an abrupt failure and the loss of a worker. Recovery needs a surviving copy of every piece of that state, such as a data-parallel replica.
- **TorchPass rolling application checkpoints retain the portable recovery foundation.** They preserve framework-defined training state frequently and asynchronously, while the application or framework remains responsible for its completeness.

A shared orchestration layer connects these mechanisms to workload health, available capacity, recovery-point freshness and destination compatibility. Operators can see what protection exists and control whether actions are manual, approval-gated or automated under policy. The objective is the least disruptive safe response available to the job.

Sources and References

1. Grattafiori, A., et al. (Meta). "The Llama 3 Herd of Models." arXiv:2407.21783, 2024.
<https://arxiv.org/abs/2407.21783>
2. Kokolis, A., et al. (Meta). "Revisiting Reliability in Large-Scale Machine Learning Research Clusters." IEEE HPCA 2025; arXiv:2410.21680. <https://arxiv.org/abs/2410.21680>
3. Amazon Web Services. "Checkpointless training on Amazon SageMaker HyperPod: Production-scale training with faster fault recovery." AWS Machine Learning Blog, 15 December 2025. <https://aws.amazon.com/blogs/machine-learning/checkpointless-training-on-amazon-sagemaker-hyperpod-production-scale-training-with-faster-fault-recovery>
4. SchedMD. Slurm documentation: "Preemption" ([preempt.html](https://slurm.schedmd.com/preempt.html)), "Generic Resource (GRES) Scheduling" ([gres.html](https://slurm.schedmd.com/gres.html)) and `slurm.conf` (`GraceTime`, `PreemptParameters`). Preemption mode `SUSPEND` keeps the job's memory and generic resources allocated; `GraceTime`: "immediately sent `SIGCONT` and `SIGTERM` ... followed by the `SIGCONT`, `SIGTERM` and `SIGKILL` signal sequence." <https://slurm.schedmd.com/preempt.html>
5. Kubernetes documentation: "Jobs" (suspending a Job), "Pod Lifecycle" (termination of Pods) and "Safely Drain a Node." "When you suspend a Job, any running Pods ... will be terminated with a `SIGTERM` signal"; `SIGTERM` with a grace period (default 30 seconds), then `SIGKILL`. <https://kubernetes.io/docs/concepts/workloads/controllers/job/>
6. Kueue documentation: "Preemption," "Cluster Queue" (`reclaimWithinCohort`) and the Workload API. Preemption "is the process of evicting one or more admitted Workloads"; admission gated through the Job's `.spec.suspend`; no checkpoint mechanism. <https://kueue.sigs.k8s.io/docs/concepts/preemption/>
7. Volcano documentation, "Actions" (preempt within a queue; reclaim across queues); NVIDIA Run:ai, "How the Scheduler Works" (consolidation by rescheduling preemptible workloads). <https://run-ai-docs.nvidia.com/saas/platform-management/runai-scheduler/scheduling/how-the-scheduler-works>
8. Amazon Web Services documentation: "Spot Instance interruption notices." <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-instance-termination-notices.html>
9. Google Cloud. "Spot VMs," section "Preemption process." <https://cloud.google.com/compute/docs/instances/spot>
10. Together AI. "Introducing preemptible compute: the same compute, half the price." 10 September 2026. A maximum five-minute `SIGTERM`-to-removal window in which the workload must checkpoint and exit.
<https://www.together.ai/blog/introducing-preemptible-compute-the-same-compute-half-the-price>
11. NVIDIA GPU Operator documentation, "GPU Driver Upgrades." Controlled driver upgrades, including waiting for selected jobs, deleting GPU-using pods and optional fallback node draining.
<https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-driver-upgrades.html>
12. Choudhury, A., et al. (Meta). "MAST: Global Scheduling of ML Training across Geo-Distributed Datacenters at Hyperscale." USENIX OSDI 2024. "Before preempting a low-priority workload for a high-priority one, it also saves a checkpoint for later restoration"; "each workload gets preempted once" on average.
<https://www.usenix.org/conference/osdi24/presentation/choudhury>
13. Qian, K., et al. "Alibaba HPN: A Data Center Network for Large Language Model Training." ACM SIGCOMM 2024. DOI: 10.1145/3651890.3672265. Section 2.3 and Figure 4 describe typical checkpoint intervals of two to four hours for four representative production large-language-model workloads.

14. Stoyanov, R., Spišáková, V., Ramos, J., et al. "CRIUgpu: Transparent Checkpointing of GPU-Accelerated Workloads." arXiv:2502.16631, 2025. <https://arxiv.org/abs/2502.16631>
15. Shukla, D., Sivathanu, M., et al. (Microsoft). "Singularity: Planet-Scale, Preemptive and Elastic Scheduling of AI Workloads." arXiv:2202.07848, 2022. <https://arxiv.org/abs/2202.07848>
16. NVIDIA Dynamo documentation, v1.4.2. "Snapshotting GPU Workers." <https://docs.dynamo.nvidia.com/dynamo/v1.4.2/knowledge-base/kubernetes/kubernetes-operator/snapshot>
17. Cedana. "Does multi-node checkpointing ship today?" 24 August 2026. <https://cedana.com/blog/cedana-multi-node-checkpointing>
18. Amazon Web Services. "Using elastic training in Amazon SageMaker HyperPod." Amazon SageMaker AI Developer Guide. <https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-eks-elastic-training.html>
19. Hu, Q., Ye, Z., Wang, Z., et al. "Characterization of Large Language Model Development in the Datacenter." USENIX NSDI 2024. <https://www.usenix.org/conference/nsdi24/presentation/hu>
20. Amazon Web Services. "Training plans utilization for SageMaker training jobs." Amazon SageMaker AI Developer Guide. <https://docs.aws.amazon.com/sagemaker/latest/dg/training-plan-utilization-for-training-jobs.html>
21. Kueue. "Topology-Aware Scheduling." https://kueue.sigs.k8s.io/docs/concepts/topology_aware_scheduling/
22. Google Cloud. "Handle GPU host maintenance events." Compute Engine documentation. <https://cloud.google.com/compute/docs/gpus/gpu-host-maintenance>
23. NVIDIA. cuda-checkpoint repository README, accessed September 2026. <https://github.com/NVIDIA/cuda-checkpoint>
24. NVIDIA/cuda-checkpoint. GitHub Issue #45, opened 10 February 2026. <https://github.com/NVIDIA/cuda-checkpoint/issues/45>
25. Cao, J., Kerr, G., Arya, K., Cooperman, G. "Transparent Checkpoint-Restart over InfiniBand." ACM HPDC 2014. <https://arxiv.org/abs/1312.3938>
26. Ghodsian, F., Clayton, J., Dinan, J., Reiss, O. (NVIDIA). "How NVIDIA NVLink 6 Delivers Multi-Layer Resiliency for AI Factories." NVIDIA Technical Blog, 15 September 2026. <https://developer.nvidia.com/blog/how-nvidia-nvlink-6-delivers-multi-layer-resiliency-for-ai-factories/>
27. CRIU project. "CUDA plugin" README. <https://github.com/checkpoint-restore/criu/tree/criu-dev/plugins/cuda>
28. Young, J. W. "A first order approximation to the optimum checkpoint interval." Communications of the ACM 17(9): 530–531, 1974. <https://doi.org/10.1145/361147.361115>
29. PyTorch. Distributed Checkpoint (DCP) documentation and tutorials. https://docs.pytorch.org/docs/stable/distributed_checkpoint.html and https://docs.pytorch.org/tutorials/recipes/distributed_checkpoint_recipe.html