PyTorch CONFERENCE 2025

CRAFT: A Pytorch-based Protocol for Composing Fault Tolerance Techniques

Fucheng Warren Zhu, Jewon Im

Introduction/Motivation

Distributed training follows the Bulk Synchronous Parallel (BSP) model that tightly couples all hardware, leading to cascading, costly failures when any component fails. While checkpoint-restart approaches incur prohibitive overhead, numerous fault tolerance (FT) solutions exist—TorchElastic, TorchFT, Prime, NVIDIA-Resiliency-Ext, Oobleck, Tenplex, Unicron—each offering valuable techniques such as parallelism reconfiguration, constant batch size guarantees, and straggler detection. However, their incompatible APIs and system designs prevent practical composition of these optimizations.

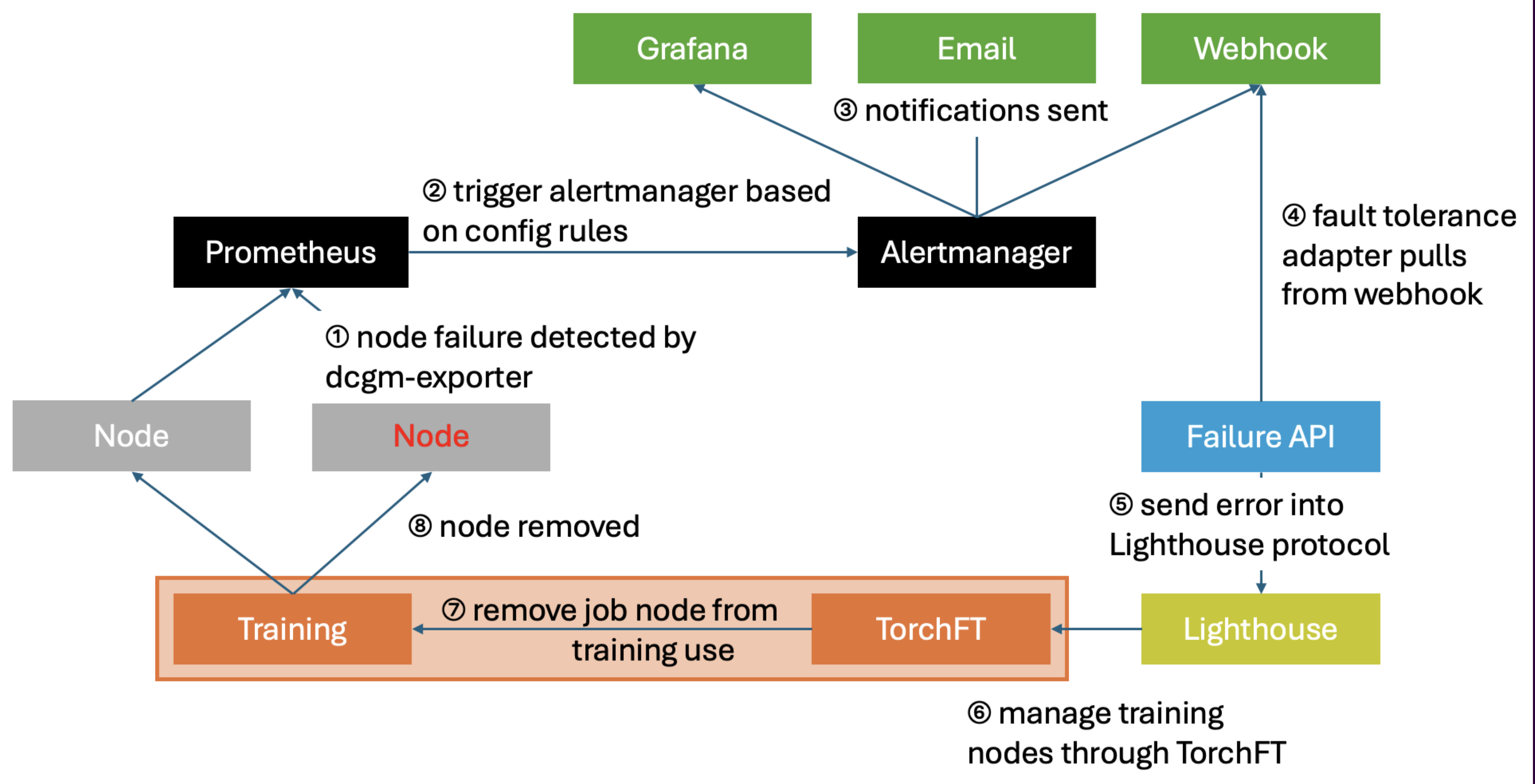
The scale of this problem is well documented. GPT-4.5 pretraining identified FT as the biggest obstacle to scaling. LLaMA-3 training encountered more than 100 faults. The BLOOM project lost 1–2 GPUs per week across 400 GPUs, costing ~1.5 hours per incident. ByteDance's Megascale, running on 10,000+ GPUs, required custom mechanisms to maintain efficiency. Hardware unreliability, expensive resources, and labor-intensive recovery together form a critical bottleneck.

Approac

CRAFT addresses this fragmentation by providing a **common protocol** that enables composing diverse fault tolerance techniques. We decompose the problem into two standardized paths:

1. **Passive Path (Training Integration):** Leverages PyTorch Lightning's lifecycle hooks as integration points, enabling FT services to operate as Lightning callbacks with consistent APIs. This allows reconfiguration only at safe computation boundaries such as iteration start/end or `optimizer.step()`.
2. **Active Path (Event-Driven):** Uses a lightweight HTTP server that routes webhooks from monitoring processes, enabling immediate interventions (e.g., aborting an NCCL all-reduce hang). This architecture is simple, extensible, and compatible with existing monitoring systems such as Prometheus.

By standardizing communication, CRAFT allows detection and response mechanisms to be swapped in and out while preserving consistent interfaces. Composability is thus achieved.


Grafana
Email
Webhook
③ notifications sent
② trigger alertmanager based on config rules
Prometheus
Alertmanager
④ fault tolerance adapter pulls from webhook
① node failure detected by dcgm-exporter
Node
Node
Failure API
⑧ node removed
⑤ send error into Lighthouse protocol
Training
⑦ remove job node from training use
TorchFT
Lighthouse
⑥ manage training nodes through TorchFT

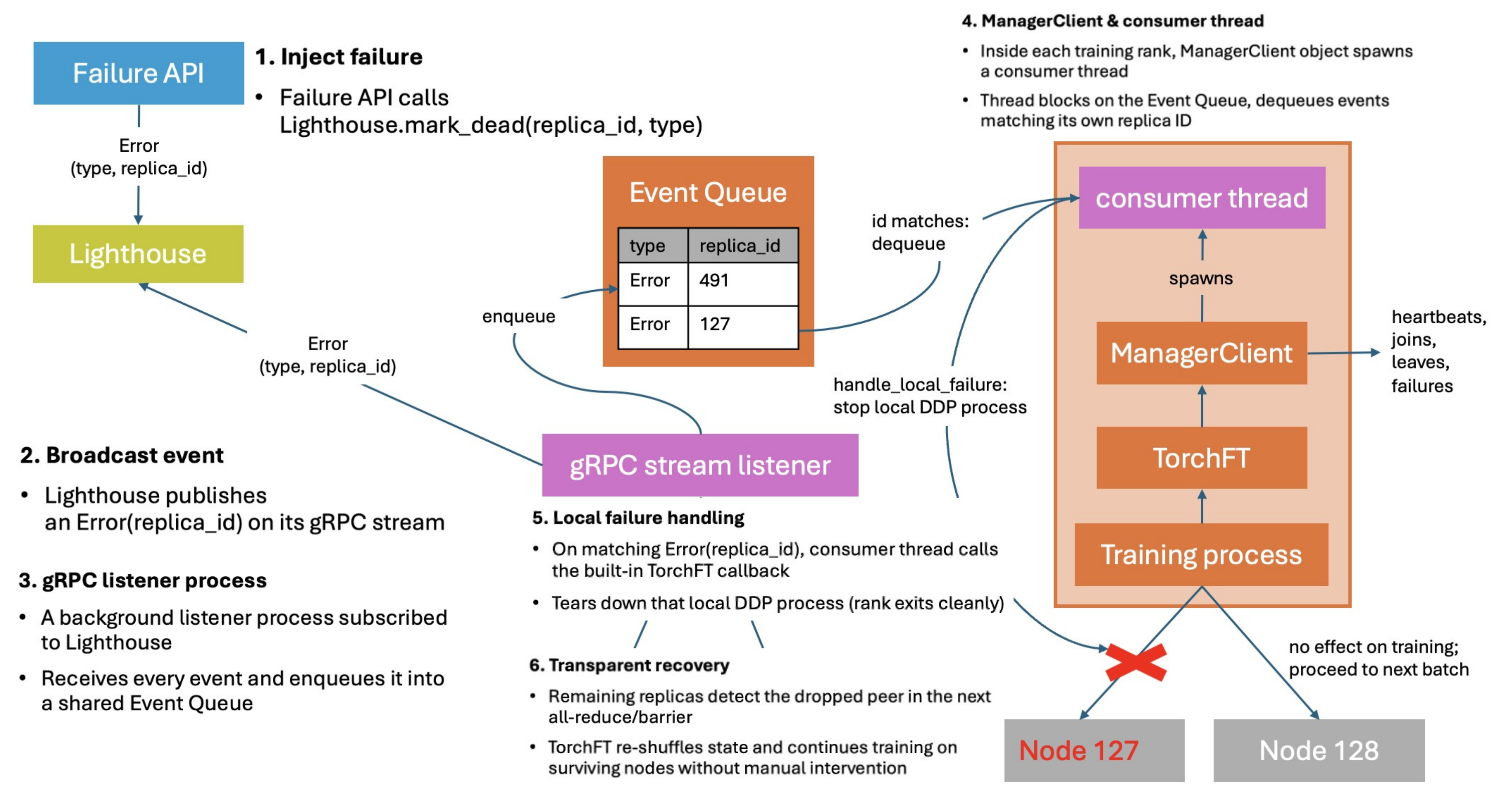
Implementation

We implemented CRAFT in NeMo by integrating TorchFT as the passive component with Prometheus alert-based reconfiguration for active monitoring. Current work extends the protocol to compose additional techniques into TorchTitan and NeMo, including:

- Straggler detection
- Topology optimization
- Parallelism reconfiguration

CRAFT unifies fault detection, communication, and response through a lightweight protocol. The architecture shows how node failures are detected, broadcast via the Failure API and Lighthouse, and transparently recovered by TorchFT callbacks during training.

Architectur



1. Inject failure
- Failure API calls Lighthouse.mark_dead(replica_id, type)

2. Broadcast event
- Lighthouse publishes an Error(replica_id) on its gRPC stream

3. gRPC listener process
- A background listener process subscribed to Lighthouse
- Receives every event and enqueues it into a shared Event Queue

4. ManagerClient & consumer thread
- Inside each training rank, ManagerClient object spawns a consumer thread
- Thread blocks on the Event Queue, dequeues events matching its own replica ID

5. Local failure handling
- On matching Error(replica_id), consumer thread calls the built-in TorchFT callback
- Tears down that local DDP process (rank exits cleanly)

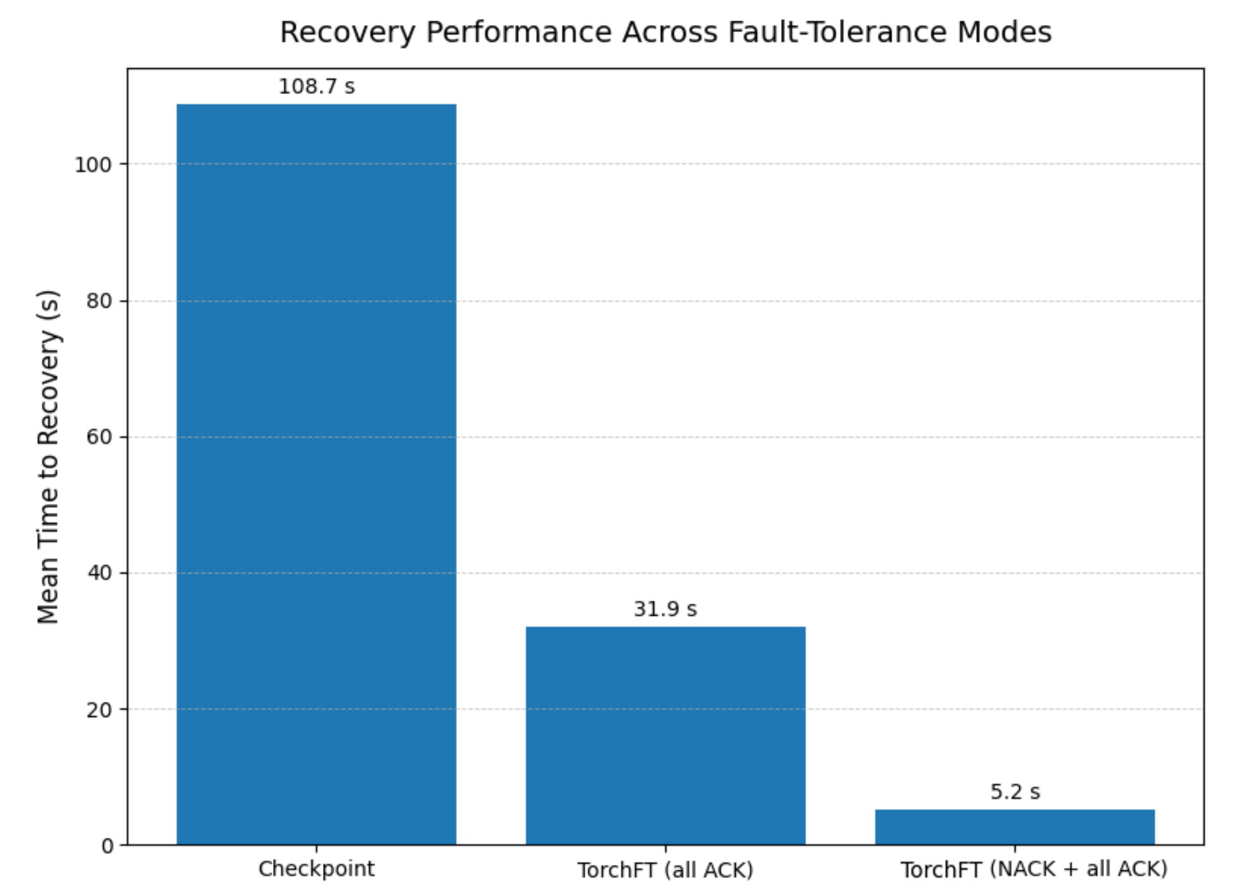
6. Transparent recovery
- Remaining replicas detect the dropped peer in the next all-reduce/barrier
- TorchFT re-shuffles state and continues training on surviving nodes without manual intervention

Results

CRAFT dominates:
- 6.1x faster than vanilla TorchFT
- ≈21x faster than checkpoint recovery.

Regular TorchFT removes the heavy checkpoint overhead but still pays a 30 s NCCL timeout before failure is detected

Classic checkpointing is the slowest because recovery can't start until the next periodic save (50-round interval in this run) and must reload model weights



Results are averaged over 20 runs, using Llama-3 8B (FSDP-4) on two GPUs; larger models will push the checkpoint bar even higher.

Conclusion & Future

CRAFT transforms fault tolerance from siloed, monolithic solutions into composable services. By standardizing interfaces and communication, it enables researchers to mix-and-match techniques without reimplementation, accelerating adoption of robust fault tolerance in large-scale ML training.

Current tests are limited to simulated environments. Active path operation depends on integrating with functioning monitoring systems. Future steps include deployment on live datacenters, comparison against Meta's TorchFT recovery setups, and extending the architecture beyond recovery into job scheduling and orchestration with direct node lifecycle management.

References