Towards Building a Flexible, Efficient and Resilient Training with Adaptive Checkpointing on AMD GPU platforms

Pratik Mishra
AMD

September 16, 2025
Hyatt Regency Santa Clara, CA
SNIA Developer Conference (SDC) 2025

AMD
together we advance_

Agenda

- AI Systems Glossary 101s
- Infrastructure, Reliability, and Foundation Model Training
- Fault-Tolerance Tax
- DeepSpeed Universal Checkpointing (UCP): *Collaboration with UIUC (Prof. Minjia Zhang)*
- Conclusion
- Copyrights and Disclaimer

Disclaimer: *Please refer to the Copyrights and Disclaimer in the presentation. We have tried to cite most relevant sources. We (the authors and associated organization) owe no responsibility towards the content's accuracy or claims, and they should be viewed as personal viewpoints/opinions to cater open discussions.*

AI Training Infra Reliability 101: Metrics

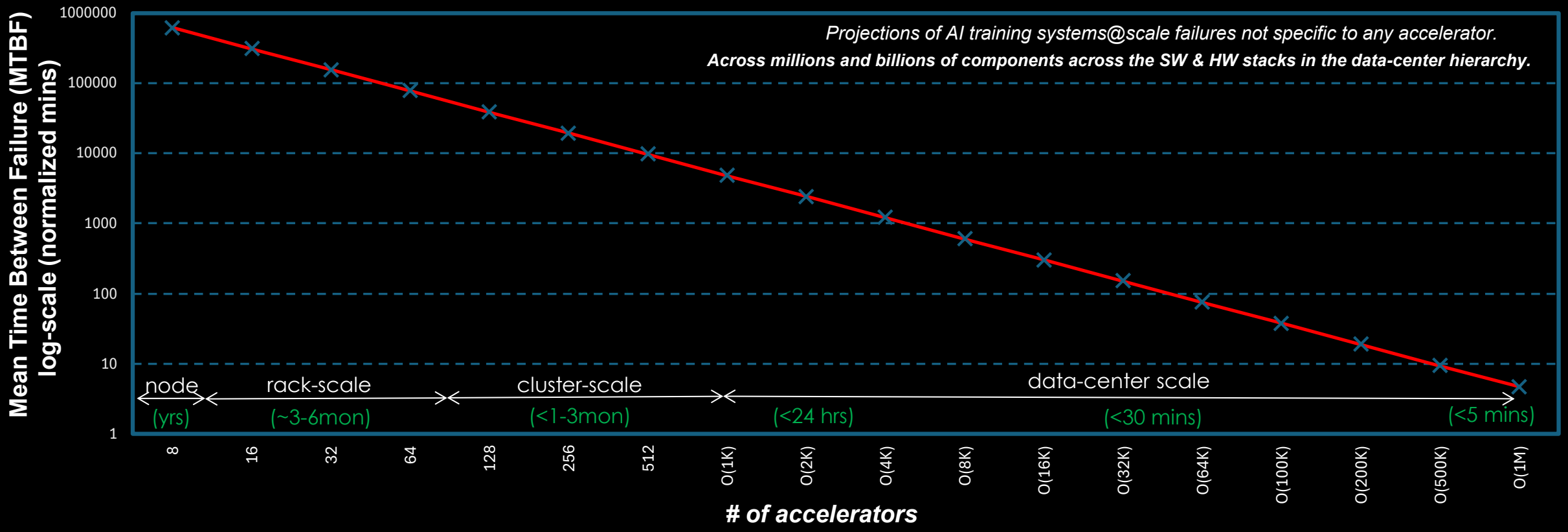
SDC'24

- **Training Goodput** = Actual progress made / total time
- **Model FLOPs Utilization (MFU)** = FLOPs a model utilizes/ peak HW FLOPs available
- **Mean Time Between Failures (MTBF)** = total time / # of failures
- **Effective Training Time Ratio (ETTR)** = actual training time / total time

Achieving high training goodput and maximizing model FLOPs utilization to improve the Effective Training Time Ratio remains a significant and ongoing challenge.

Failures and Training Efficiency?

Reliability and Training Efficiency @scale



$$MTBF \propto 1/(no.of\ accelerators)$$

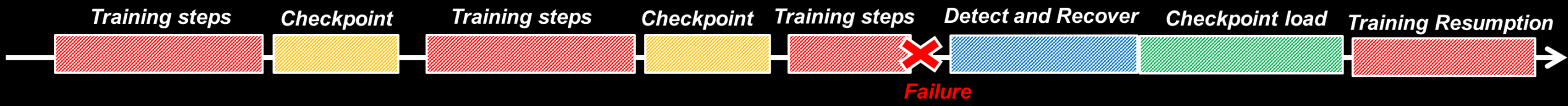
With growing scale of AI deployments, the MTBF decreases significantly.

Therefore, resiliency is the core for achieving Training efficiency and increasing Training Goodput and ETTR.

Fault Tolerance, Training Efficiency and Checkpointing

- Fault-tolerance, resiliency, and recovery are of utmost importance for Training Efficiency metrics (discussed earlier)

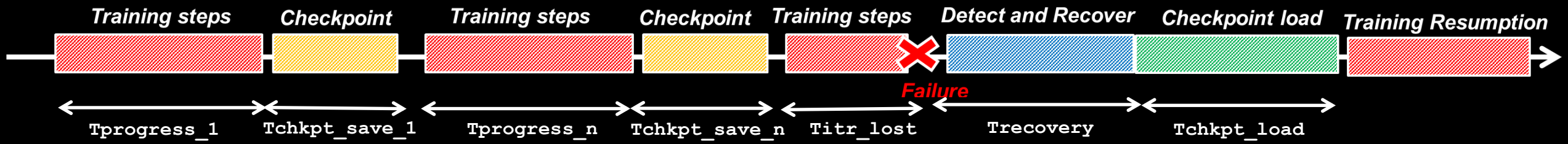
- Storage community's poster AI use-case: ***Checkpointing***



- Critical fault-tolerance mechanism for periodically persisting training snapshots to enable recovery via rollbacks in the event of failure
  - *Also: Hardware refresh, Resource re-balancing, post-training, concurrent evaluation, increase accuracy, etc.*

With scale and every-lowering MTBFs, the checkpointing frequency, size, and complexity increases significantly; imposing heavy data-center tax (GPU underutilization).

Fault Tolerance Tax: Checkpointing



$$FT_{overhead} = T_{chkpt\_save} + T_{itr\_lost} + T_{recovery} + T_{chkpt\_load}$$

$$ETTR = (1 - FT_{overhead})$$

- Achieving optimal ETTR @ data-center scale is **"real"** challenge
  - *Without optimization, systems may spend more time managing failures than actual training*
  - **Trade-off**: Excessive checkpoints increases data-center tax & infrequent increases risks (cost)
  - **Data-center tax**: compute, network, storage

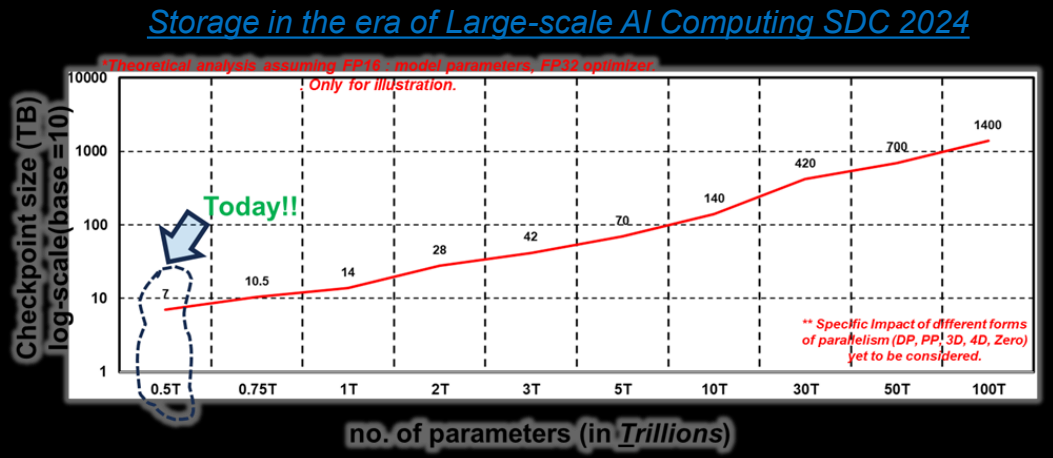
Therefore, to achieve optimal* ETTR (+goodput) *it is essential for reliability mechanisms to strike the balance of performance, scalability, and cost-effectiveness.

Checkpointing (save)

- **C**heckpoint (save) = **S**erialization + **P**ersistence

- **S**erialization
  - Create tensor compatible file(s)/object(s)
  - {GPU states + CPU states + Metadata}

- **P**ersistence
  - Write chkpt file(s)/object(s) to persistent storage
  - Size, access pattern, layout → depends on the parallelism + framework
  - For details: *Storage in the era of Large-scale AI Computing, SDC 2024*

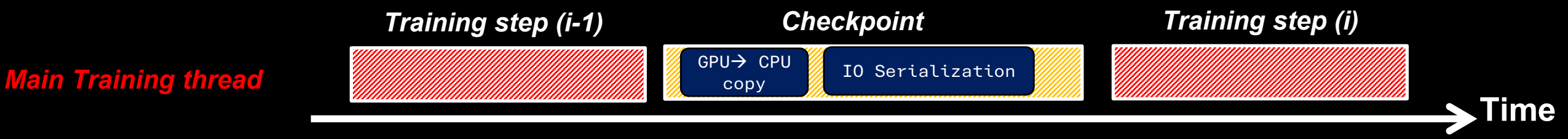


T_{chkpt_save}

With growing model sizes, checkpointing frequency, size grows exponentially gets more distributed and complicated (persisting and restore).

Checkpointing (save) : Traditional

Synchronous chkpt : Main Training thread waits till checkpoint is persisted

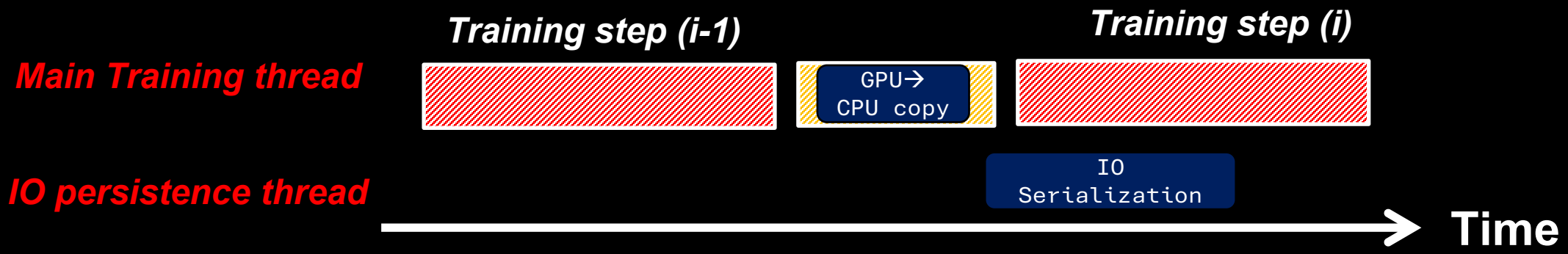


- Simple yet introduces training stalls
- GPU underutilization (stalling) + costs
- Bursty with strict SLAs

Synchronous Checkpointing poses unique throughput, latency and network with short, periodic, bursty writes which can over-subscribe the storage infrastructure and leads to GPU stalling to resume training.

Checkpointing (save) : Optimization

Asynchronous chkpt : Main Training thread is alleviated from IO persistence



- Overlaps IO and computation
- Alleviates main GPU training thread from IO critical path
- Reduces persistence pressure on storage system(s) by buffering
- CPU memory in the case of PyTorch DCP [1]
  - Though not truly async (IO verbs) !!!

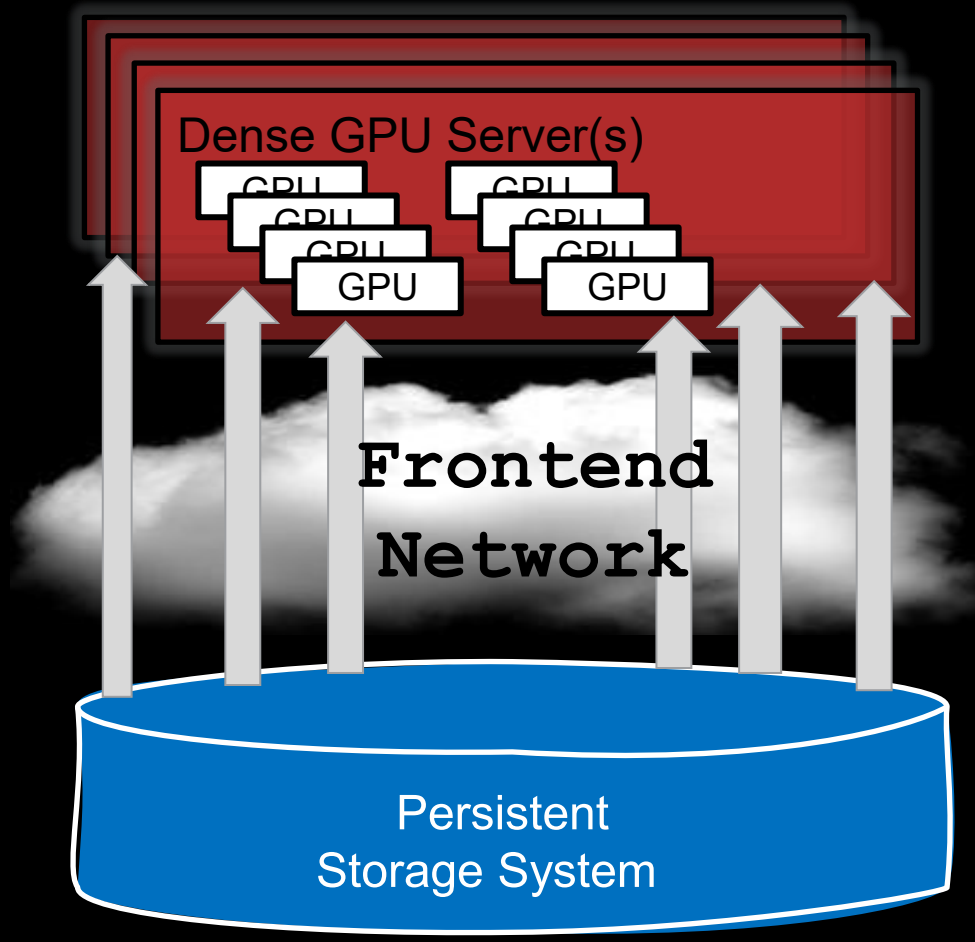
Current implementations need further optimizations to reduce @scale overheads.
For example, advanced memory + storage tiering (local NVMe → PFS → Object) to mask the IO and computation latency, while aiming to be truly "asynchronous".

[1] PyTorch Asynchronous Checkpointing: *https://pytorch.org/blog/reducing-checkpointing-times/*

Checkpointing (loads)

T_{chkpt_load}

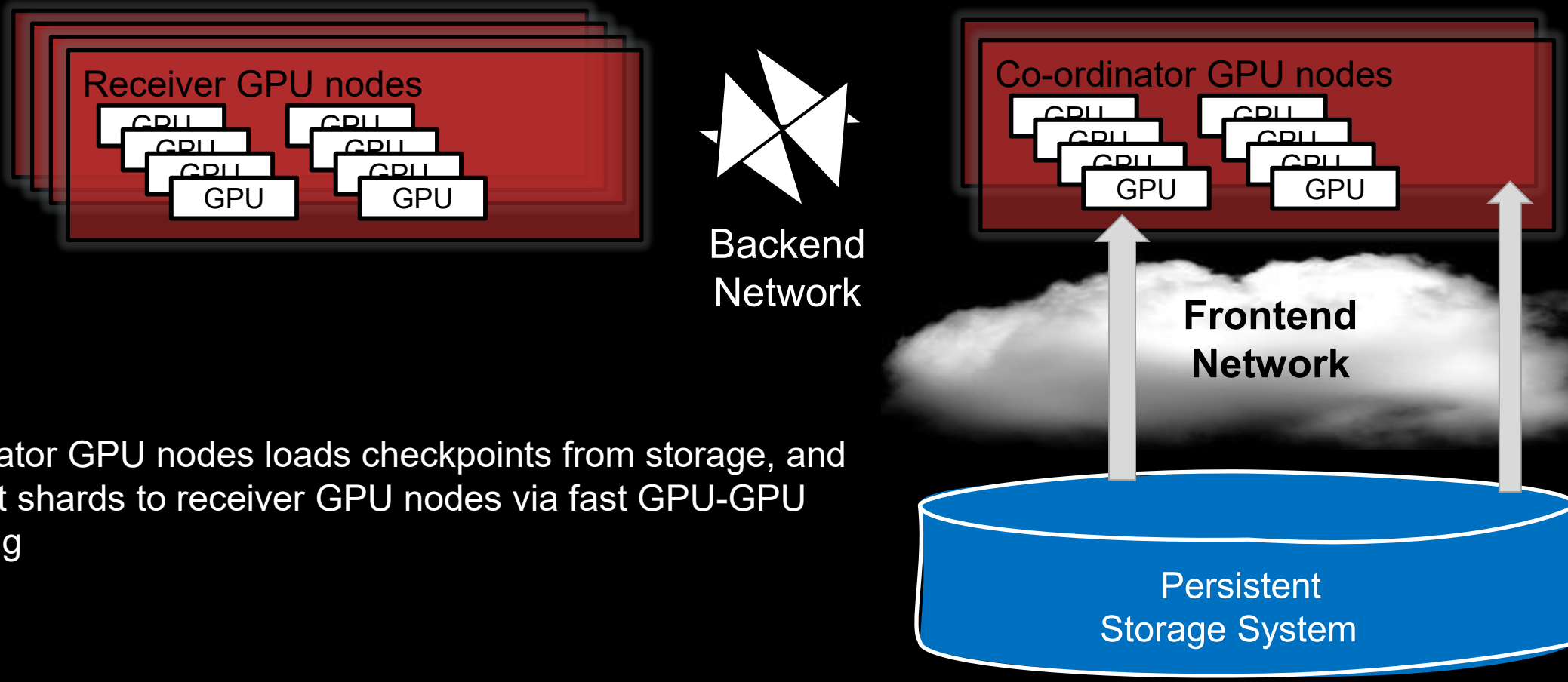
- **C**heckpoint (load) = **L**oading + **D**eserialization
- **L**oading
  - Reading chkpt file(s)/object(s) from storage systems
- **D**eserialization
  - Unpickle object(s) – metadata + CPU → GPU amenable structures
- All GPU simultaneously load their states to resume training
  - Massive IO amplification compared to save(s)
  - Also, downstream tasks – post-training, inference, etc.



Loading checkpointing is mission critical.
This leads to "real" bottlenecks across the GPU – Storage hierarchy:
throughput + tight latency + network contentions and congestions + multi-tenancy.

Checkpointing (loads) : Optimizations

Compute network-aware checkpoint loading

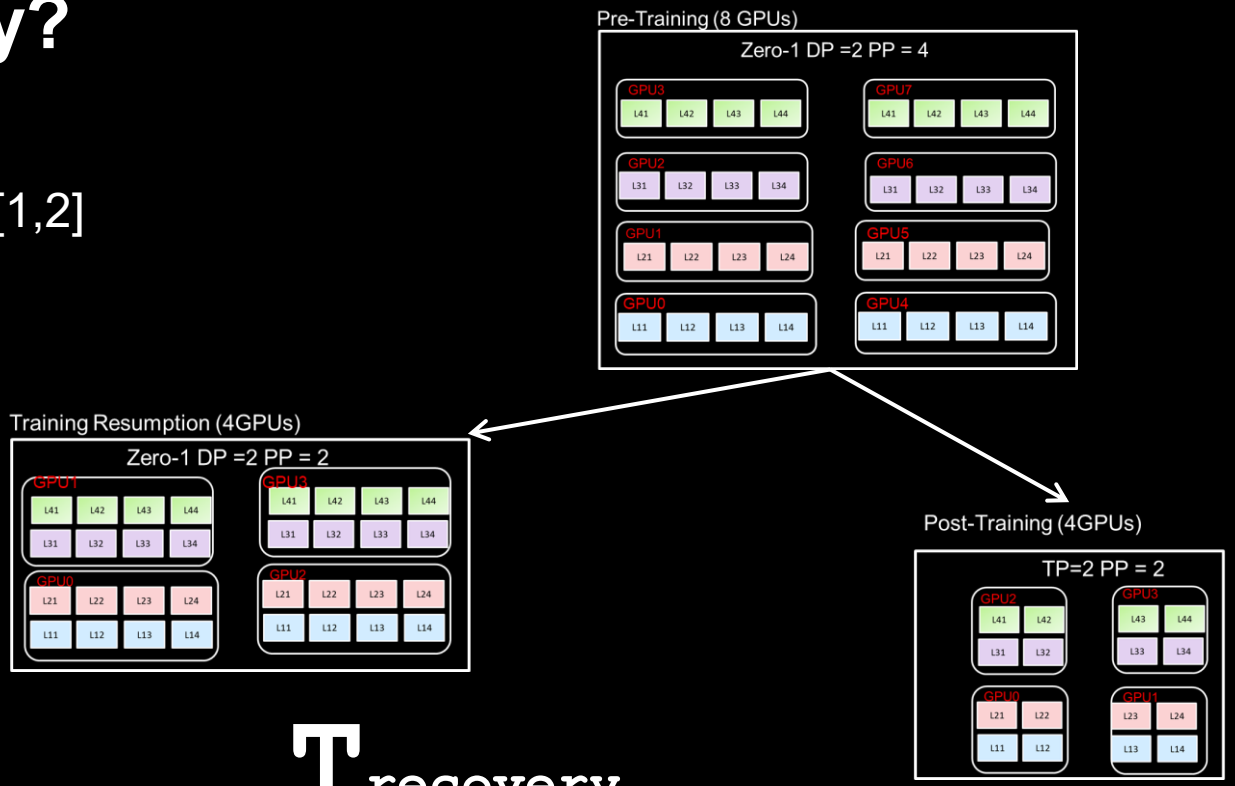


- Co-Ordinator GPU nodes loads checkpoints from storage, and broadcast shards to receiver GPU nodes via fast GPU-GPU networking

Efficient fault-tolerant checkpointing loading at scale requires GPU–storage path optimizations and topology-aware strategies to sustain robust infrastructure and high MFU.

Recovery with Flexibility + Elasticity?

- Resource rebalancing (GPU shape change) is common [1,2]
  - **Training Resumption :** reconfiguration parallelism
  - **Post-Training** : lower requirement for SFT, RL
  - **Inference** : much lower with diff. config + data-set



- Existing distributed training frameworks provide highly limited support for reconfiguring parallelism.
  - Mostly inefficient: offline, hand-written scripts, human intervention

$$T_{recovery} + T_{chkpt\_load}$$

Distributed checkpoints are tightly coupled to initial parallelism and HW configuration, resulting in GPU idle time (recovery time) during re-sharding limiting adaptability to resource elasticity.

[1] Wan, Borui, et al. "{ByteCheckpoint}: A Unified Checkpointing System for Large Foundation Model Development." *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 2025.
[2] DeepSpeed UCP: Lian, Xinyu, et al. "Universal checkpointing: Efficient and flexible checkpointing for large scale distributed training." *arXiv preprint arXiv:2406.18820* (2024). ***Accepted in USENIX ATC'25.***

Supporting flexible, efficient and resilient training on AMD GPUs with DeepSpeed Universal Checkpointing

Collaboration: Prof. Minjia Zhang (UIUC), co-creator of DeepSpeed UCP; and PhD students (Jiankun Wang and Xinyu Lian)

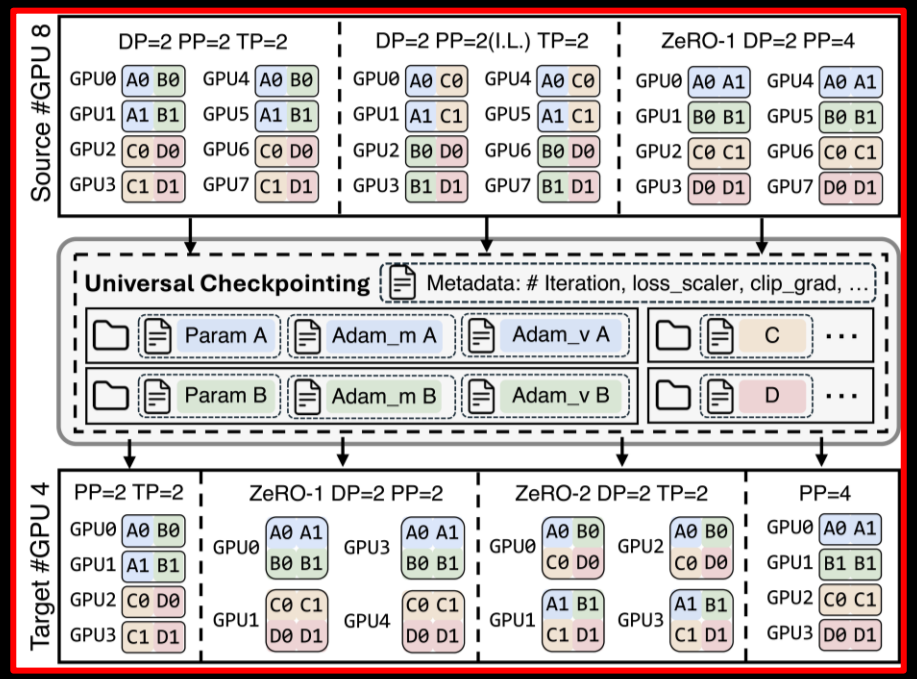
UCP : Universal Checkpointing

- Developed as a part of DeepSpeed
  - Support for commercial-scale models (BLOOM, Megatron GPT, Llama, Microsoft Phi)

- Comprehensive, flexible, and automated
  - Checkpoint re-sharding along most training parallelism techniques
    - Combinations - Zero-DP, PP, TP, DP, SP

  - Defines UCP language to support checkpoints from various frameworks (e.g. DCP)
    - Pattern matching: runtime-sharding information

DeepSpeed UCP [2]



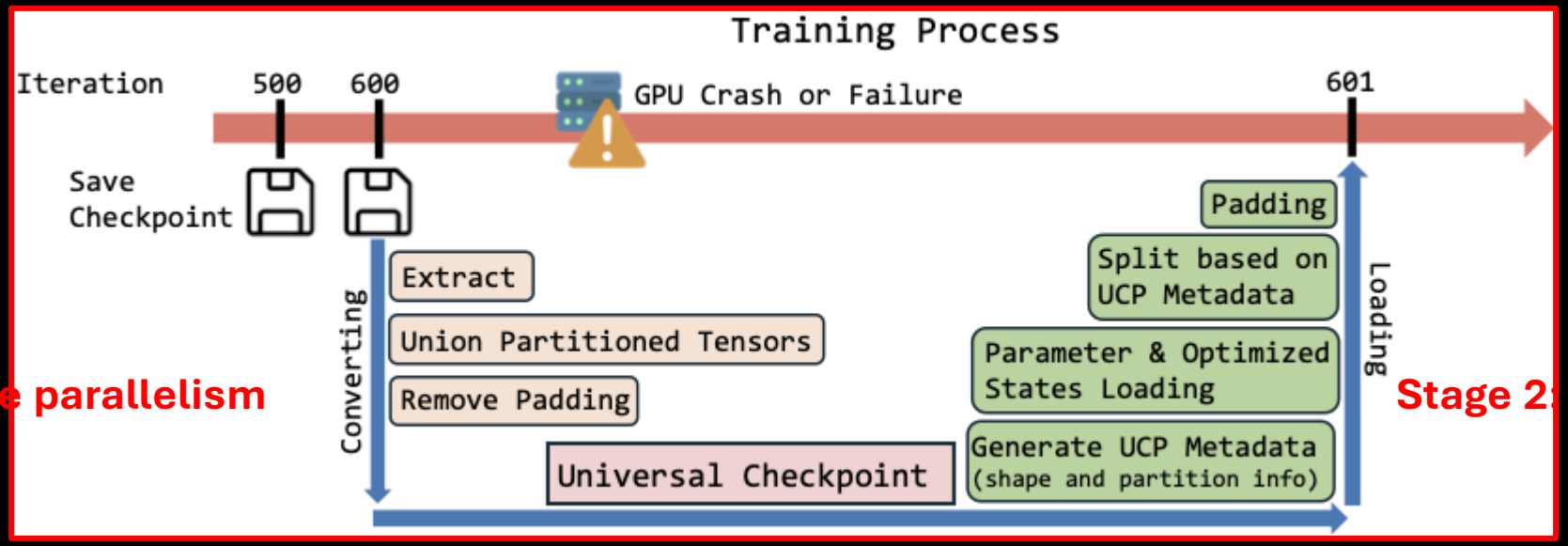
DeepSpeed UCP

[1] Github: https://github.com/deepspeedai/DeepSpeed/blob/master/blogs/deepspeed-ucp/README.md

[2] Paper: Lian, Xinyu, et al. "Universal checkpointing: Efficient and flexible checkpointing for large scale distributed training." *arXiv preprint arXiv:2406.18820* (2024). ***Accepted in USENIX ATC'25.***

UCP : 100K birds-eye view

DeepSpeed UCP 2024



Stage 1: Decouple parallelism

Stage 2: Load and Convert

From source distributed checkpoints re-create per-parameter consolidated view/ “atomic checkpoints.”

“atomic checkpoints” per parameter: Weight, Momentum, Variance.

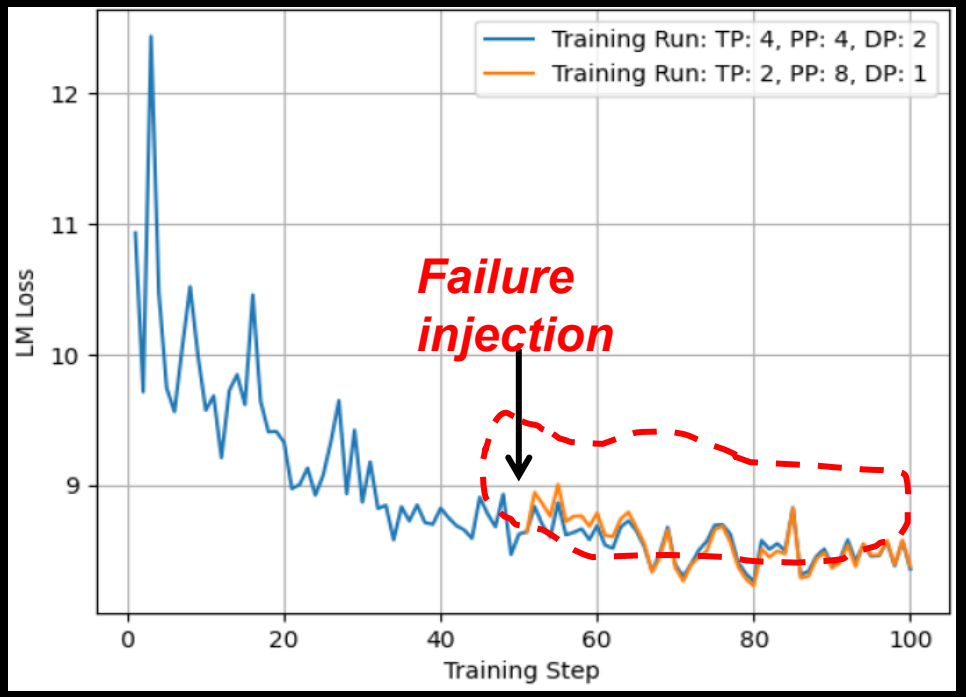
Based on UCP language pattern-matching; re-shard from atomic checkpoints to target GPU configurations.

UCP: Accuracy

Recovery from checkpoint needs to be accurate, fast and agnostic to changing parallelism patterns

Blue denotes the actual training run loss, and orange denotes the loss after checkpoint recovery with changing parallelism

Experiments over GPU clusters and remote high-performance NVMe storage system.



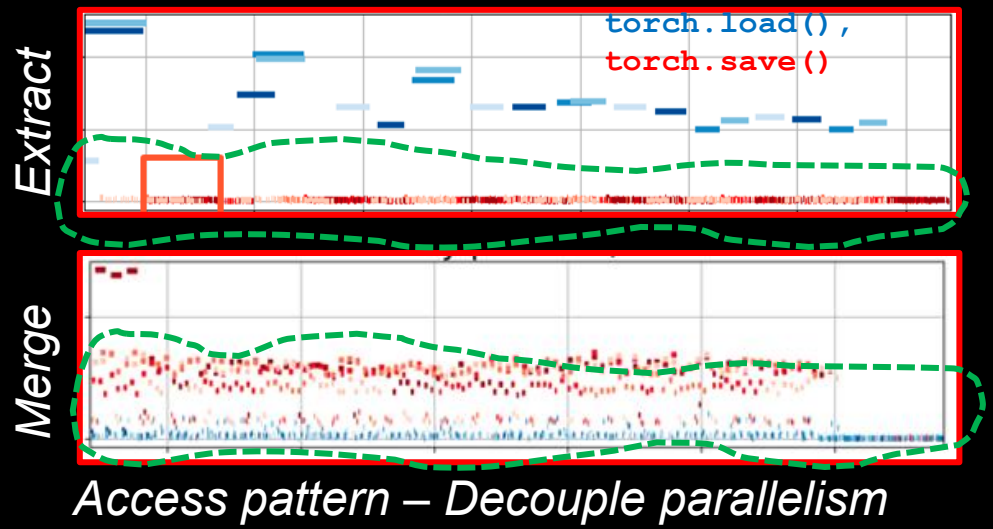
Initial Configuration (blue) : 8 MI210 nodes (32 GPUs) with TP=4, PP=4, DP=2.
Resume Training (orange) : 4 MI210 nodes (16 GPUs) with TP=2, PP=8, DP=1.

UCP enables failure recovery with resource rebalancing (GPU shape, parallelism) without compromising training accuracy.

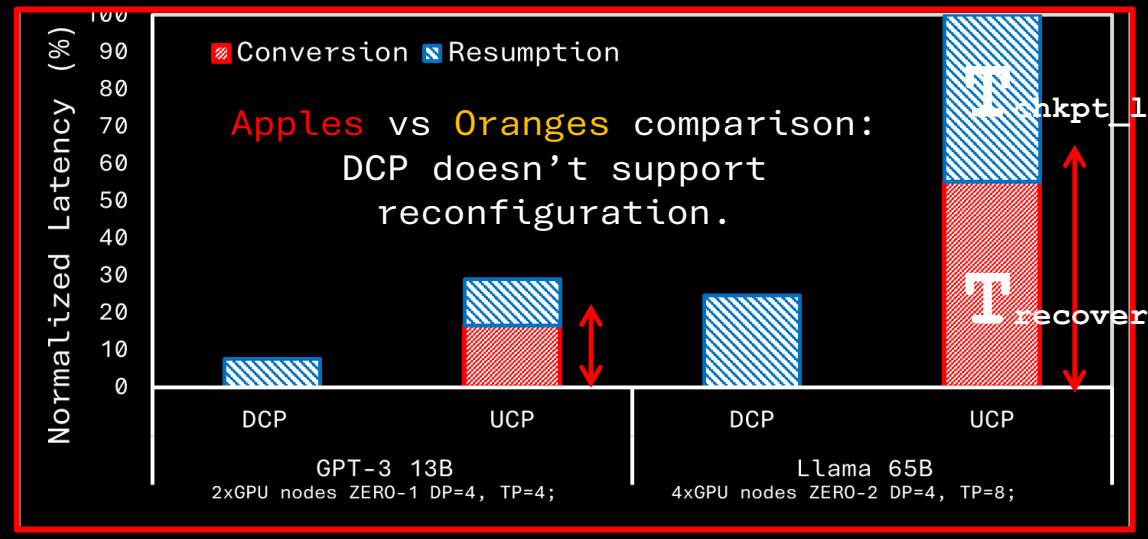
UCP: Under the Hood Analysis

- **UCP** has to do ***extra work*** compared to DCP for reconfiguration
  1) Decouple parallelism 2) Convert and Load to target GPU shapes

UCP IO volume > 4x DCP due to reconfiguration



Access pattern – Decouple parallelism



- High GPU node host-resource consumption
  - Large # of temporary intermediate files
  - Time, size and phase-varying access pattern

- GPU – remote storage BW underutilization
  - Serialization and Deseralization
  - Opportunity to exploit in-node parallelism

UCP needs to perform extra work for elastic recovery.
However, it needs adaptive optimizations to reduce recovery time cost-effectively.

UCP: Architectural Re-design

Infrastructure-aware optimizations

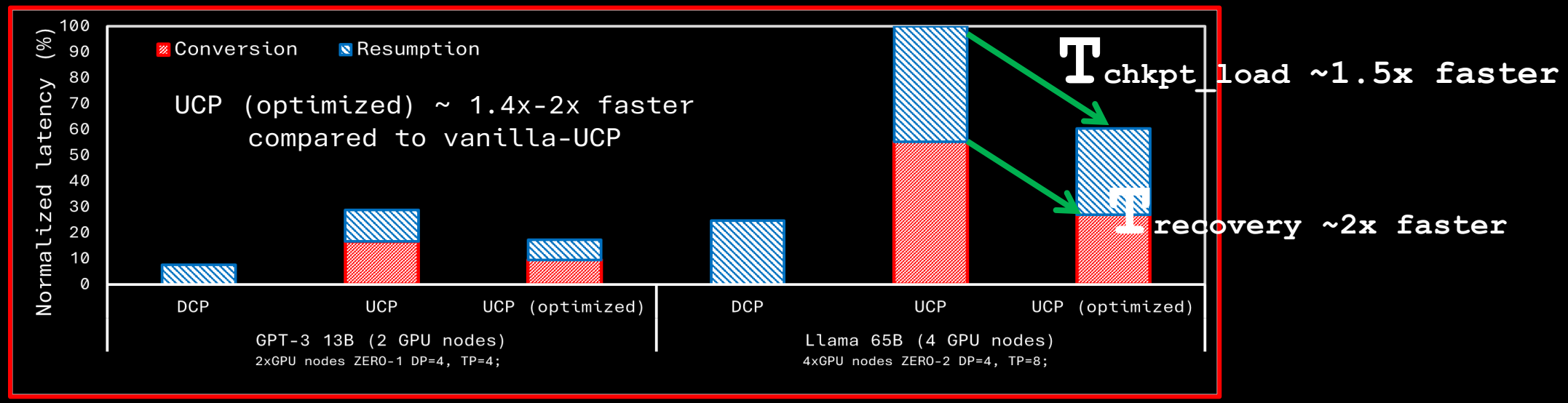
- Storage characteristic - throughput, backend (object/file), scalability analysis, etc.
- Cluster and GPU-node topology (network BW)

+ Inter/intra-node optimizations

- Dynamic, adaptive GPU-node host-resource (memory, compute) + workload-aware
- Multi-node + async Hierarchical parallelism

+ Metadata-aware optimizations

- Deserialization chkpt (.pt) file structure-aware
- mmap + offset-based dynamic loading
  Elimination of temporary file creation



IO traffic 2x reduction.

Reduction in GPU-node host resource consumption, IO traffic, etc.
Increase in GPU-Storage BW utilization.
Resulting in lower latency

UCP optimizations across the GPU–storage data path significantly reduce recovery and resumption time (+cost), improving training goodput and ETTR.

Conclusion

- ***Trend is clear***:
  - With scale and size of AI deployments, failures will be inevitable, while MTBF will keep lowering
  - Robust, scalable, and cost-effective fault-tolerance recovery and resiliency mechanisms is the core to achieve optimal ETTR and Training goodput
  - Resource rebalanced recovery is becoming common in the AI lifecycle

- Therefore*, AI Training Fault-tolerance needs to be flexible, resilient, elastic and adaptable*
  - UCP (Universal Checkpointing) seems to be promising direction for automated, flexible, resilient, and elastic AI Training. *However, it needs full-stack scalable optimizations*

Therefore, to achieve optimal* ETTR (+goodput) *it is essential for reliability and recovery mechanisms to strike the balance of performance, scalability, and cost-effectiveness to harness the full potential of GPU-accelerated AI computing.

COPYRIGHT AND DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY


AMD