Survey of Fault-tolerant LLM Training

JIANG, Zhihan

Ph.D. Student

The Chinese University of Hong Kong

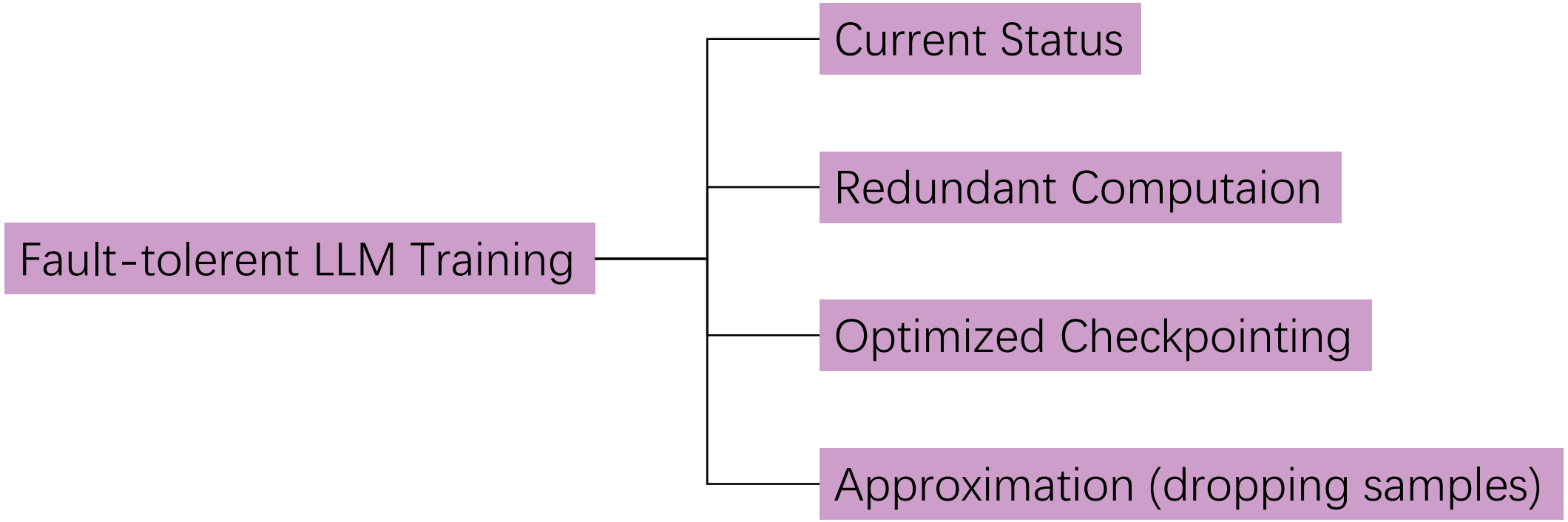
2023-11-16

HUAWEI

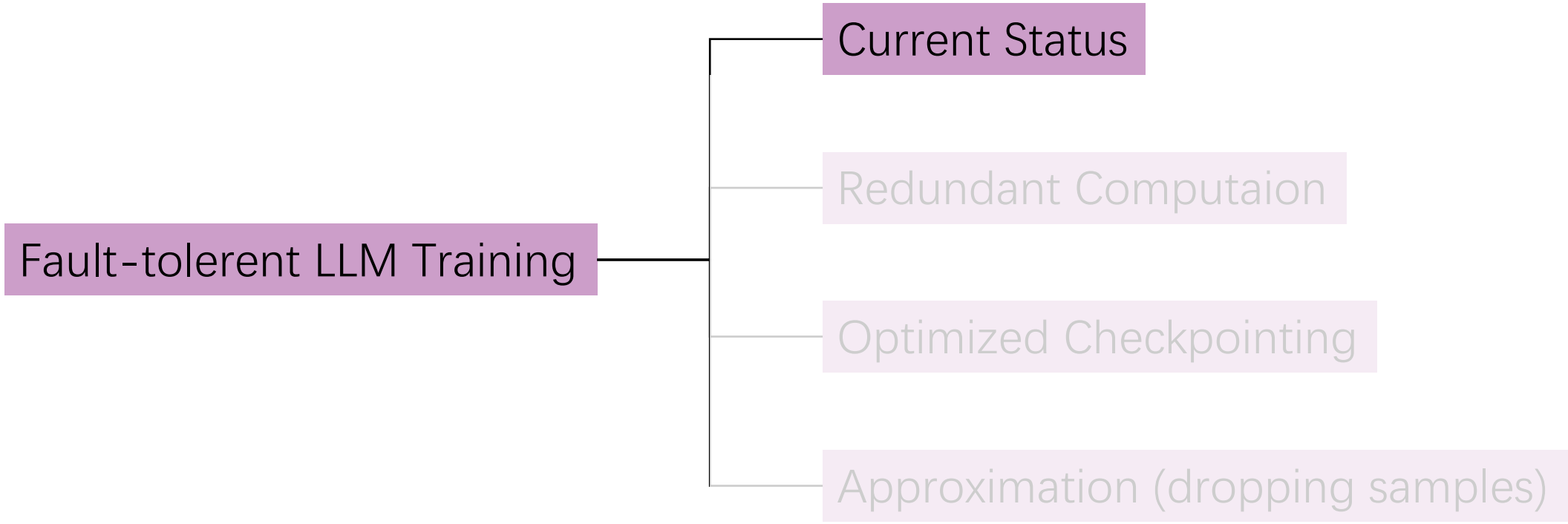
ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong

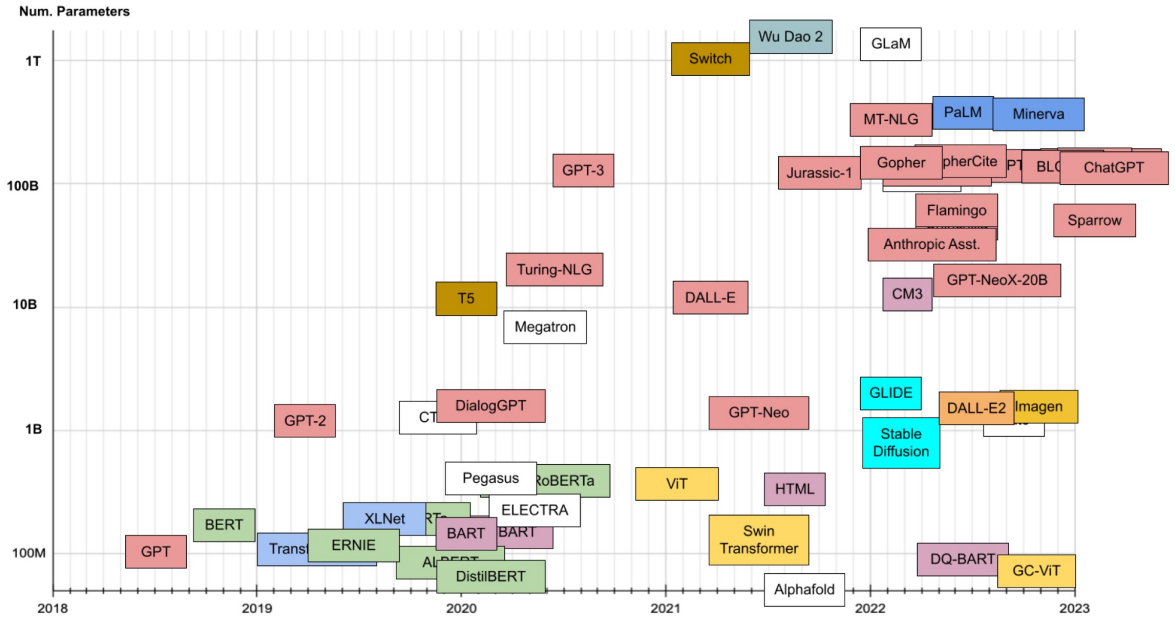
Outline


Fault-tolerent LLM Training
Current Status
Redundant Computaion
Optimized Checkpointing
Approximation (dropping samples)

Outline


Fault-tolerent LLM Training
Current Status
Redundant Computaion
Optimized Checkpointing
Approximation (dropping samples)

Why we need fault-tolerant LLM training?



The sizes of LLMs and training data are scaling up

Model	# A100-40GB	Throughput (tokens/sec)	Hours / 1B Tokens	Multi-node speedup	Price / 1B Tokens: Oracle [BM.GPU4.8]	Price / 1B Tokens: AWS [p4d.24xlarge] on-demand	Price / 1B Tokens: AWS [p4d.24xlarge] spot
GPT3-125M	8	191,693	1.45	1.00x	$35.36	$47.49	$14.24
GPT3-125M	32	739,328	0.38	3.86x	$36.67	$49.25	$14.77
GPT3-125M	128	2,471,936	0.11	12.90x	$43.87	$58.92	$17.67
GPT3-350M	8	69,427	4.00	1.00x	$97.62	$131.11	$39.33
GPT3-350M	32	269,926	1.03	3.89x	$100.44	$134.89	$40.46
GPT3-350M	128	958,464	0.29	13.81x	$113.14	$151.96	$45.58
GPT3-760M	8	57,549	4.83	1.00x	$117.77	$158.17	$47.45
GPT3-760M	32	218,317	1.27	3.79x	$124.18	$166.78	$50.03
GPT3-760M	128	696,320	0.40	12.10x	$155.74	$209.16	$62.74
GPT3-1.3B	8	38,912	7.14	1.00x	$174.18	$233.93	$70.17
GPT3-1.3B	32	152,781	1.82	3.93x	$177.45	$238.32	$71.49
GPT3-1.3B	128	561,152	0.50	14.42x	$193.25	$259.55	$77.86

The training machines and time are also increasing

the computational demand for large-scale AI models doubles approximately every 10 months

Compute trends across three eras of machine learning, 2022

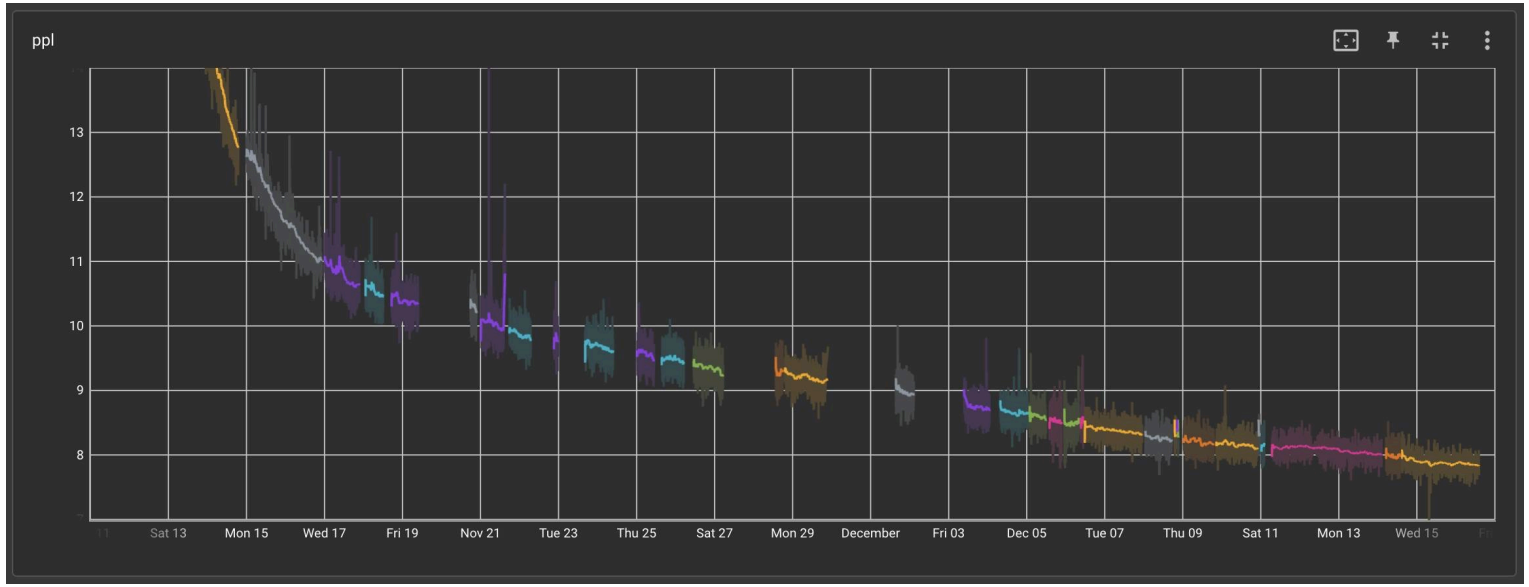
Why we need fault-tolerant LLM training?

- **As the size of the model parameters and training machines increases, the probability of failures during training also significantly increases.**

35+ manual restarts

70+ automatic restarts

100+ cycling hosts



All in all, working around infrastructure issues has dominated the last two weeks of the team's time, given that these hardware issues can take the experiment down for hours at any time of the day.

Since the sleepless night of Thanksgiving break, this past week has been filled with gradient overflow errors / loss scale hitting the minimum threshold (which was put in place to avoid underflowing) which also causes training to grind to a halt. We restarted from previous checkpoints a couple of times.

OPT: Open Pre-trained Transformer Language Models (175B, from Meta): 992 A100 80GB, 2month

Why we need fault-tolerant LLM training?

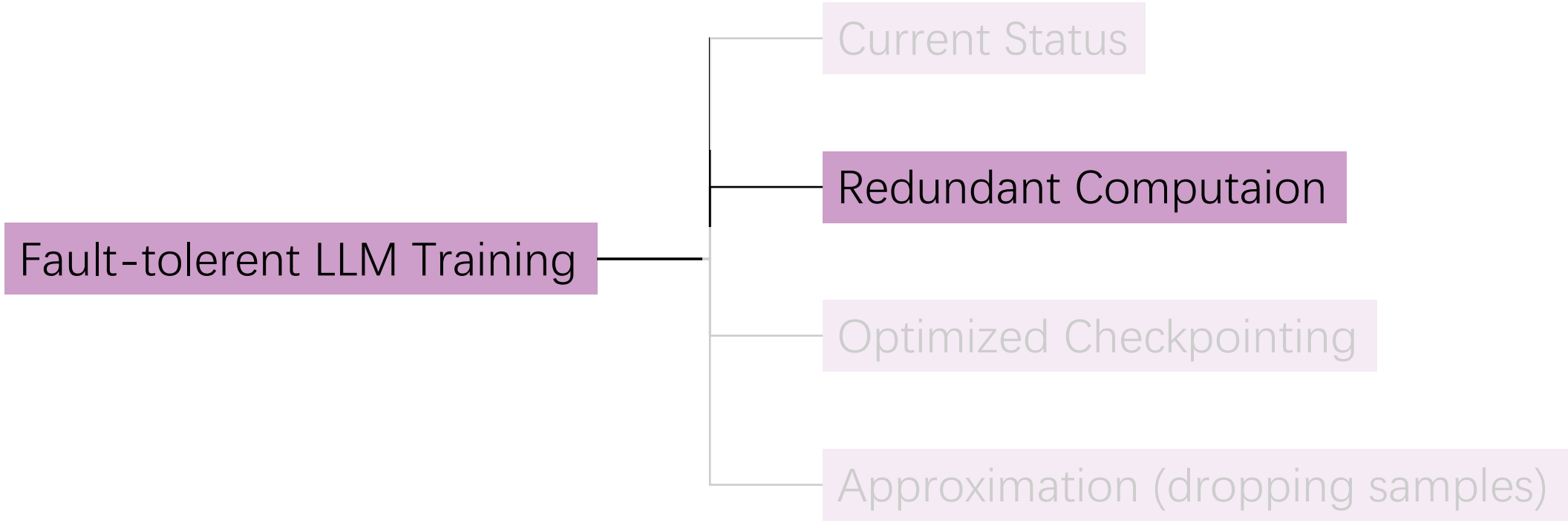
LLMs Training Task Error Statistics (From May 2023 to July 2023, running on SenseCore cluster)

Error Categorization	Number of Tasks	Root Cause
Storage Read/Write Errors	34	Due to synchronization anomalies of storage servers, significant variations in the time overhead for file storage or object storage occur across different nodes. This leads to communication waiting timeout or socket timeout in tasks.
Network Communication Errors	43	Incorrect insertion of IB network card; Uneven load distribution in RDMA traffic; Misconfigured RoCE network switch; Expired ARP cache IP; Ethernet card or link is not connected.
Node Hardware and Software Errors	66	GPU ECC errors; GPU failure; Node not ready; Insufficient shared memory; Pod sends SIGTERM signal to exit; Pod OOMKilled; Node image pull timeout; Local storage exceeding limit error.
User Code and Training Environment Errors	179	Error in creating duplicate files with the same name; Data type conversion error; Python Segmentation fault; CUDA runtime version mismatch with CUDA driver; AttributeError; torch.cuda.OutOfMemoryError; RuntimeError; ModuleNotFoundError; NameError; AssertionError; OSError.
Others	55	System hang without error output; Occasional socket timeout errors without specific issues identified during troubleshooting; Random occurrences of Pod processes with exit code -9 errors; Pod startup failures.

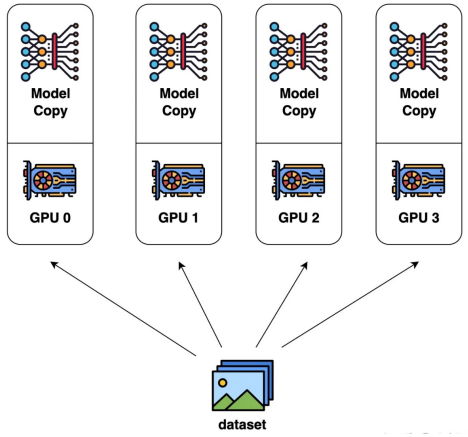
Approach	Fault Tolerant	Anomaly Detection	Checkpoint Optimization
DeepSpeed	✗	✗	✓
PyTorch Elastic	✓	✗	✗
Horovod Elastic	✓	✗	✓
Singularity	✓	✗	✓
DeepFreeze	✗	✗	✓
PAI	✓	✗	✗
ModelArts	✓	✓	✗
Azure	✗	✗	✓
TRANSOM	✓	✓	✓

Attention to the fault tolerance of large language models is increasing!

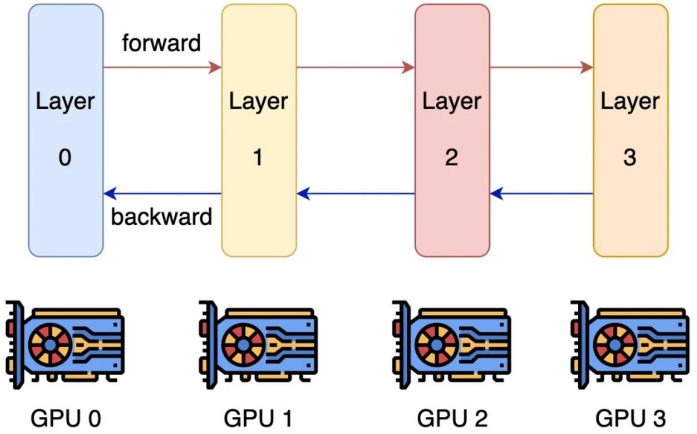
Outline


Fault-tolerent LLM Training
Current Status
Redundant Computaion
Optimized Checkpointing
Approximation (dropping samples)

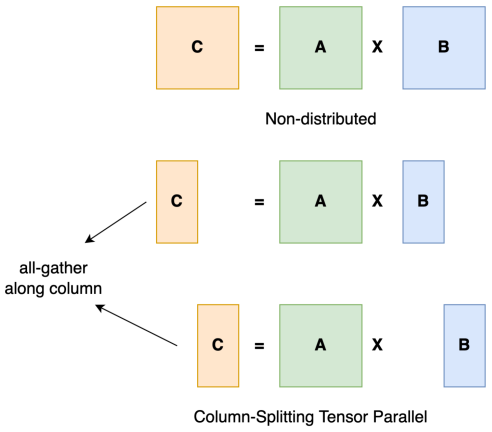
Parallel Training in large-scale DNNs

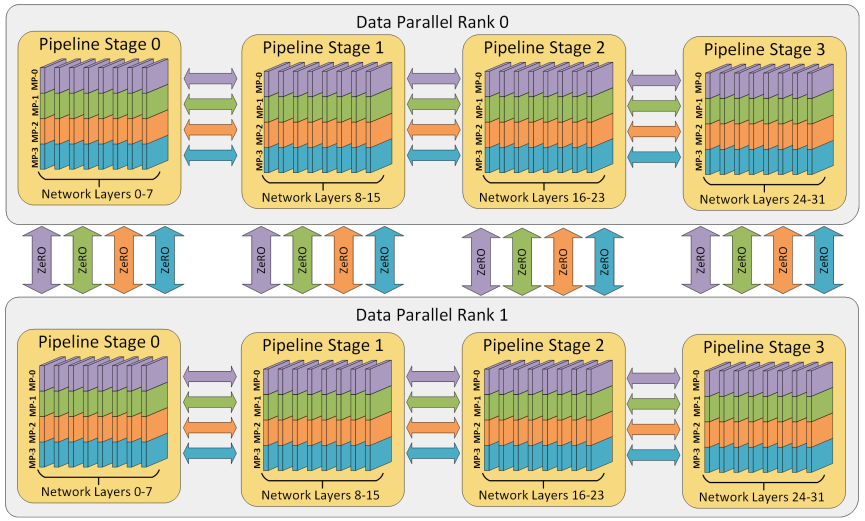


Data Parallelism

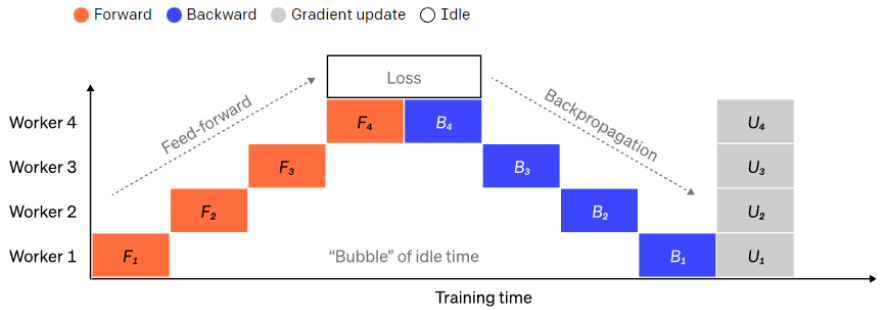


Pipeline Parallelism

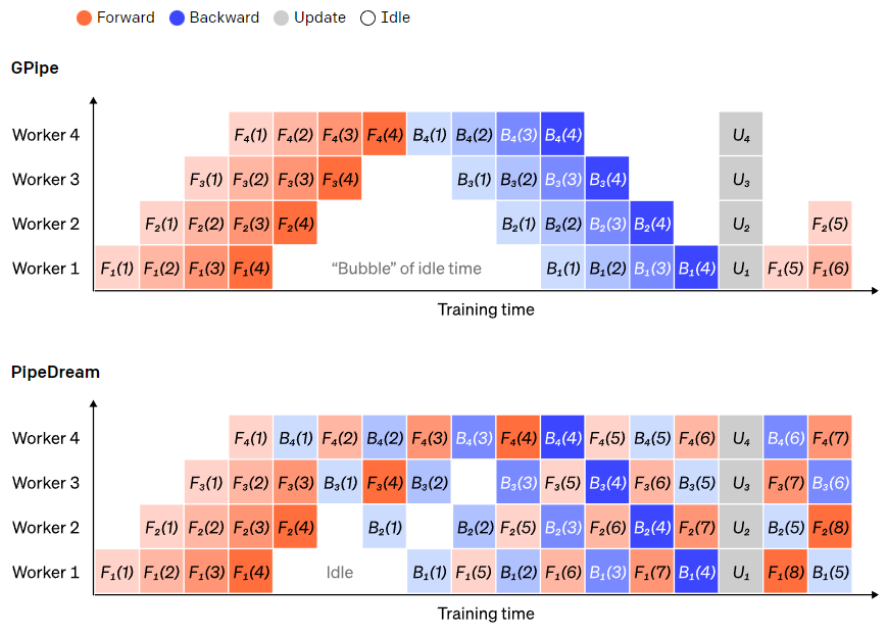




Tensor Parallelism



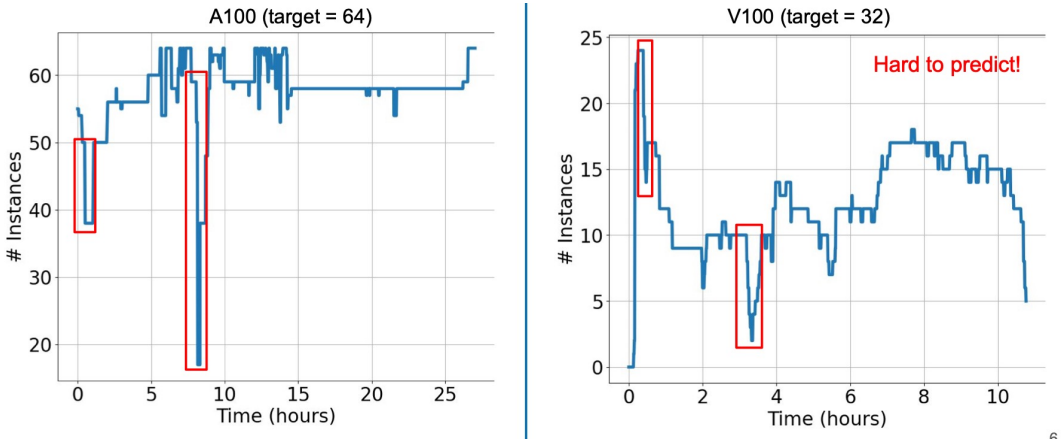
Naive Pipeline



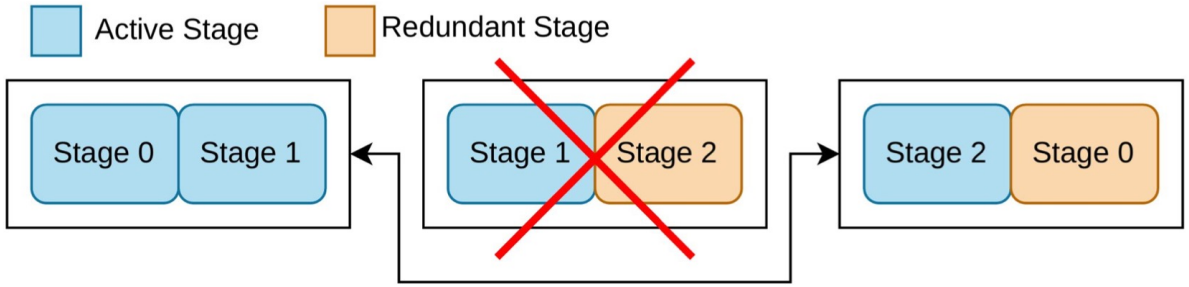
Optimized Pipeline

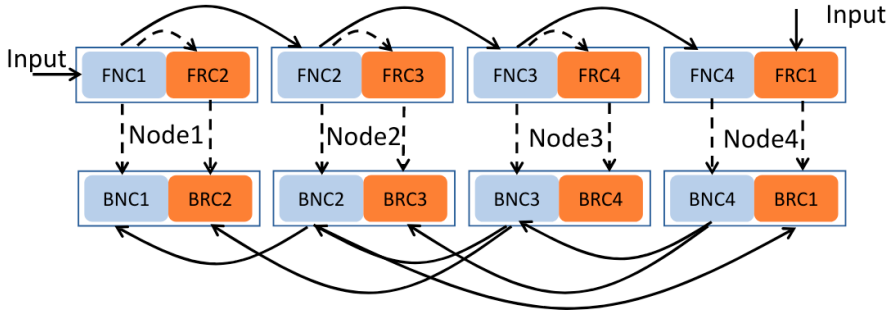
Bamboo: pipeline redundant computation

- Model sizes are increasing, the cost of training is higher
- Spot instances can lower costs, but have high failure rates
  - up to 70% cheaper
  - Preemptions can be unavoidable



- Redundant Computation (RC)
  - inspired by disk redundancies such as RAID
  - each node carries its own shard of layers and its successor's shard (exploit locality)



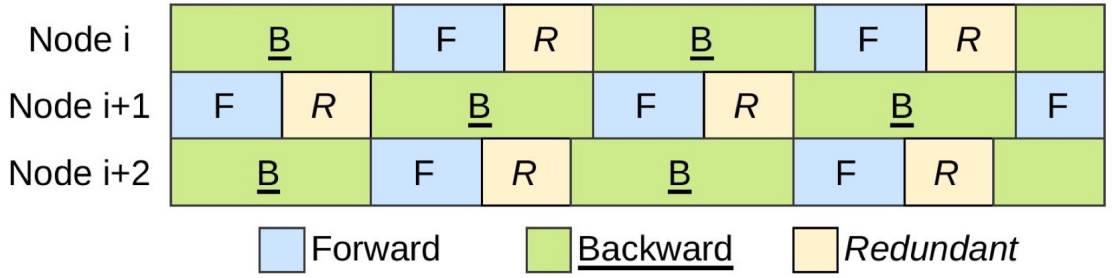


[NSDI23] Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large DNNs (from UCLA & MSR)

Bamboo: pipeline redundant computation

Challenge 1: high overhead

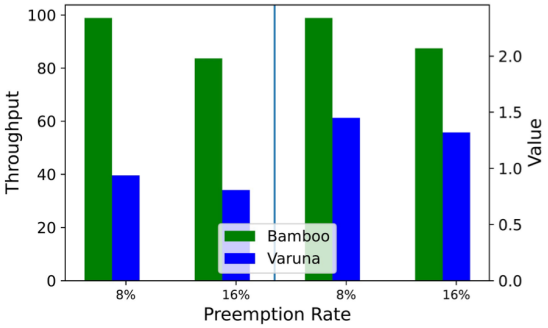
- pipeline parallelism has bubbles
- use this idle time to minimize redundancy overhead



Challenge 2: high GPU memory usage

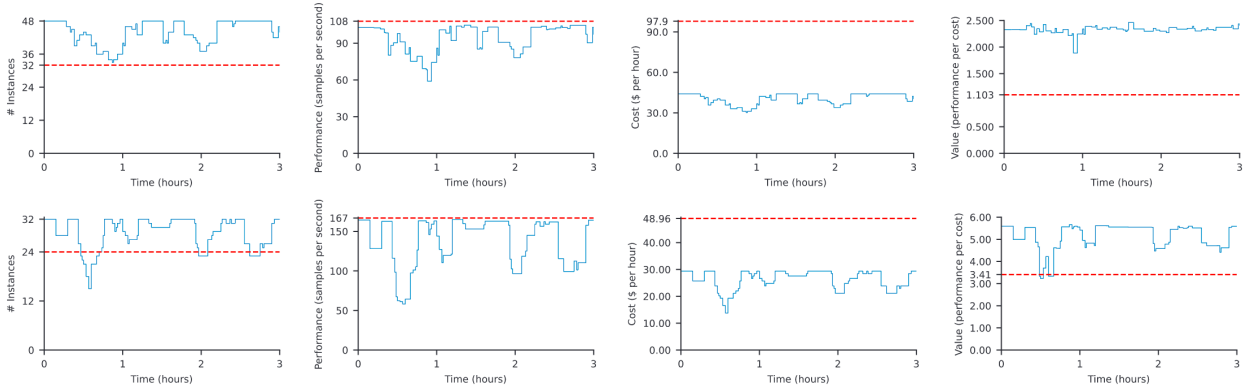
- swap out the intermediate results of each node's FRC into the node's CPU memory (offload less frequently used tensors to CPU memory)

Model	Dataset	Samples	D	P
ResNet-152 [22]	ImageNet [32]	300,000	4	12
VGG-19 [63]	ImageNet [32]	1,000,000	4	6
AlexNet [32]	ImageNet [32]	1,000,000	4	6
GNMT-16 [68]	WMT16 EN-De	200,000	4	6
BERT-Large [15]	Wikicorpus En [15]	2,500,000	4	12
GPT-2 [49]	Wikicorpus En [15]	500,000	4	12



Challenge 3: consecutive failure

- make consecutive nodes in each pipeline come from different zones



(a) Trace (b) Training Throughput (c) Monetary Cost (d) Value

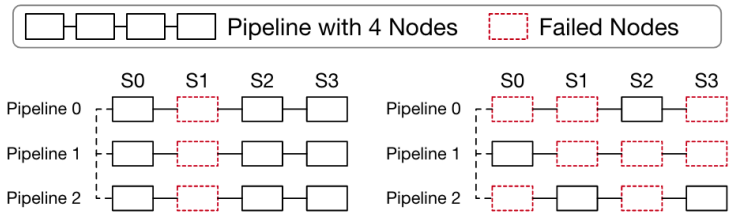
compared to on-demand instances

[NSDI23] Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large DNNs (from UCLA & MSR)

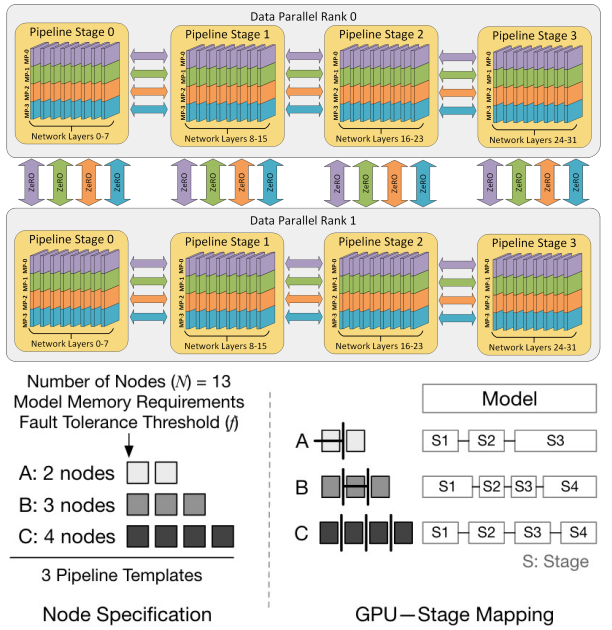
Oobleck: scheduling pipeline templates

- Scenarios: **hybrid-parallel training (DP + PP)**
- Model state redundancy in DP is free redundancy

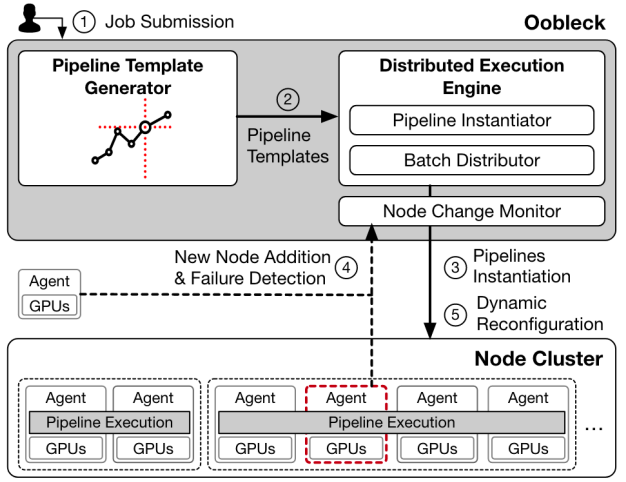
Can we leverage this redundancy in DP for reliability?



(a) *Failures happen to nodes with the same stage (S1 lost, not recoverable)* **(b)** *Failures in random places (all stages alive, recoverable)*



(a) *Generating pipeline templates (§4.1)*



The core design: **pipeline templates**

- a pipeline specification that defines how many nodes should be assigned to a pipeline, how many stages to create, and how to map model layers in stages to GPUs.

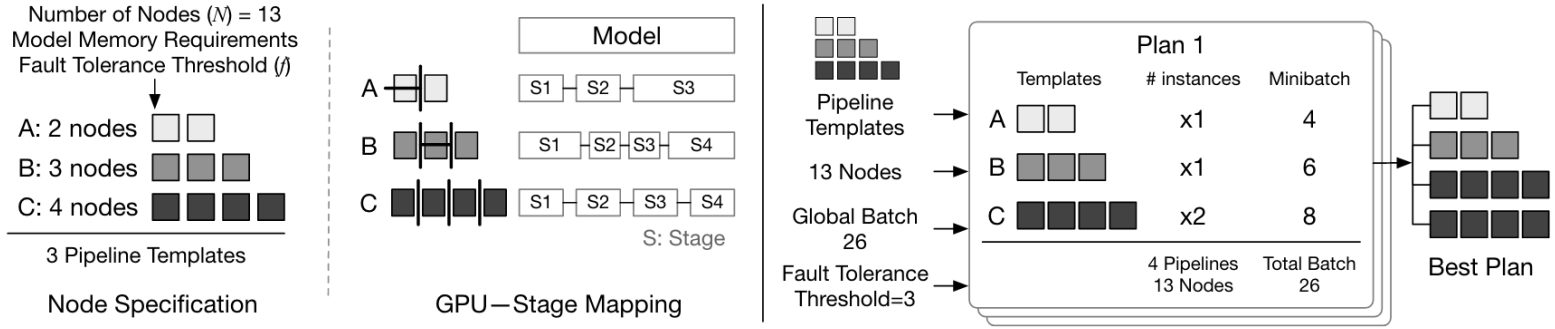
Decoupling planning (pipeline template generation) from execution (pipeline instantiation) enables fast failure recovery;

[SOSP23] Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates (from umich & Amazon)

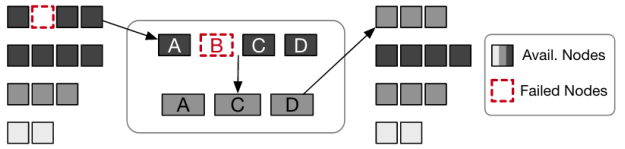
Oobleck: scheduling pipeline templates

Oobleck Planning Algorithm

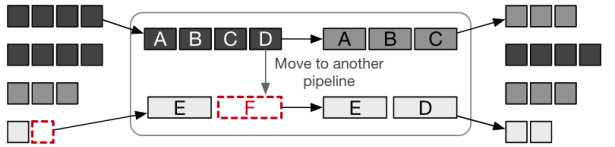
- Generating pipeline templates
  - Node specification
  - GPU–Stage mapping
- Pipeline instantiation
  - Enumerating all instantiation options
  - Calculating throughput with batch distribution
- Dynamic Reconfiguration
  - Pipeline re-instantiation
  - Batch redistribution
- Other designs
  - Model synchronization between heterogeneous pipelines in a layer granularity



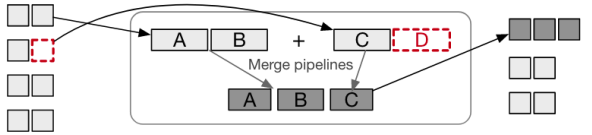
(a) *Generating pipeline templates (§4.1)* **(b)** *Pipeline instantiation (§4.2)*



(a) *A node failure in a 4-node pipeline. We have a 3-node pipeline template, thus a new pipeline with 3 nodes is instantiated, which replaces the existing one.*



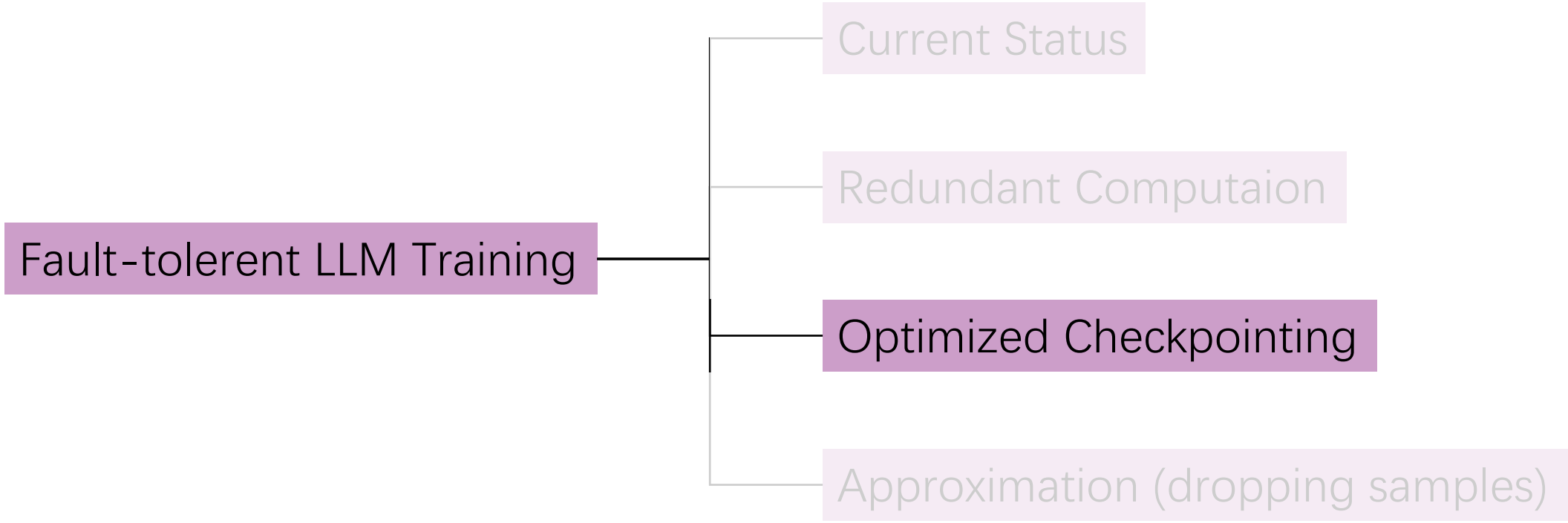
(b) *A node failure in a 2-node pipeline. Since there is no template for one node, it gets another node from another pipeline to keep the 2-node pipeline. Two affected pipelines reinstantiate or reconfigure themselves.*



(c) *A node failure in a 2-node pipeline. Because it cannot borrow a node from any other pipeline, it is merged with another pipeline.*

[SOSP23] Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates (from umich & Amazon)

Outline

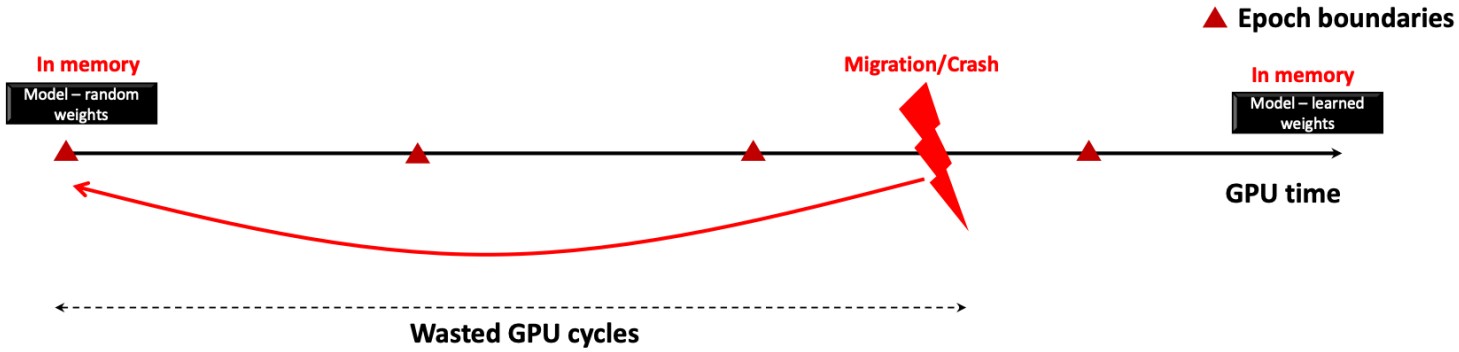

Fault-tolerent LLM Training
Current Status
Redundant Computaion
Optimized Checkpointing
Approximation (dropping samples)

CheckFreq: fine-grained and pipelined checkpointing

Three main challenges in large-scale DNNs checkpointing:

- **Checkpoint stalls**: how to minimize cost of checnkpoint?
- **Checkpointing frequency**: how offten to checkpoint?
- **Data invariant**: how to resume correctly?

Any interruption can wipe out the model parameters learned so far in memory, restarting this expensive process!



Checkpoint of a 128B LLM: ~ 1.536 TB

Trade-off between low-overhead and high frequency of checkpointing

[FAST21] CheckFreq: Frequent, Fine-Grained DNN Checkpointing (from UT Austin & MSR)

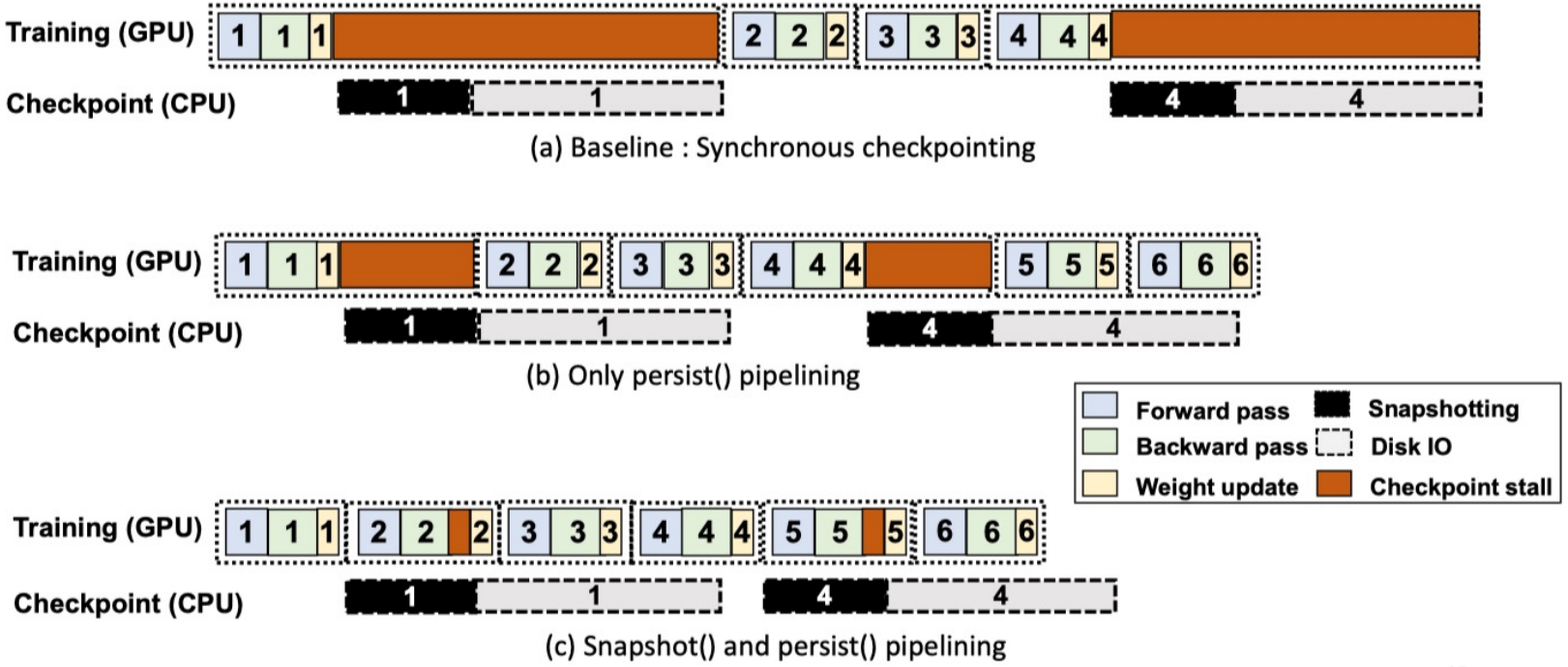
CheckFreq: fine-grained and pipelined checkpointing

Checkpoint stalls: how to minimize cost of checkpoint?

2-phase DNN-aware checkpointing

Low checkpoint stalls

- Synchronous checkpointing introduces checkpoint stalls => Runtime overhead
- Low-cost checkpointing mechanism that is split into two pipelined operations:
  - Snapshot() : Serialize and copy into a user-space buffer
  - Persist() : Write out the serialized contents to disk



[FAST21] CheckFreq: Frequent, Fine-Grained DNN Checkpointing (from UT Austin & MSR)

CheckFreq: fine-grained and pipelined checkpointing

Checkpointing frequency: how often to checkpoint?

automatically profiles metrics and determines frequency

Systematic online profiling

Initial checkpointing frequency

Iteration time | Time for weight update | Time for GPU snapshot() | Time for CPU snapshot()

Adaptive rate tuning

Manages interference from other jobs

Available disk throughput | Checkpoint size | Peak GPU memory util | Total GPU memory

Data invariant: how to resume correctly?

epoch seeded psuedo-random transformations

Resumable data iterator

Maintain data invariant

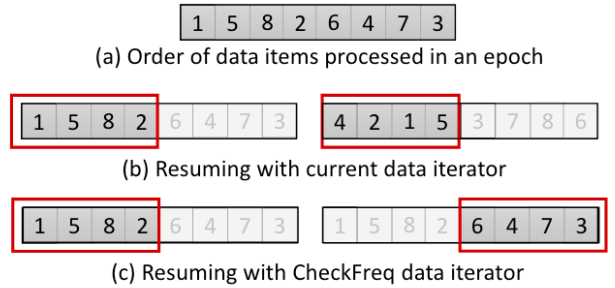


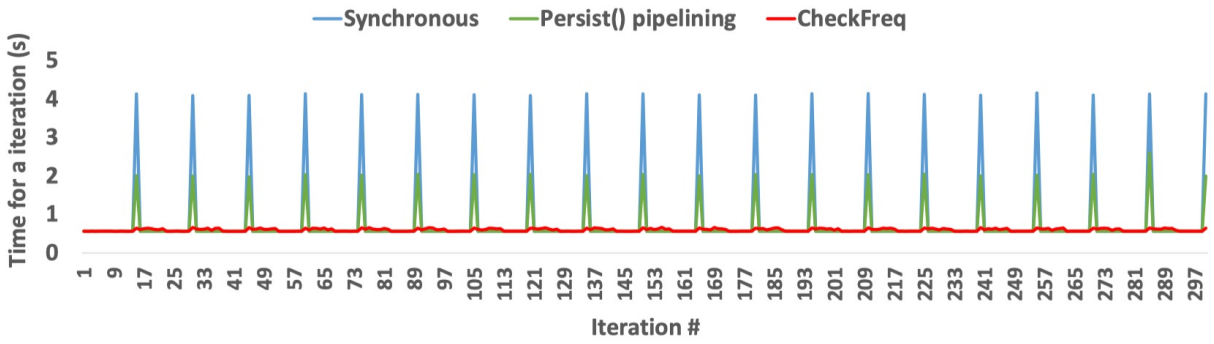
Figure 5: **Resuming iterator state**. When iterator state is not resumable, an epoch might miss data items when job is interrupted (items 3,6,7 are missed in b). CheckFreq (c) ensures that training resumes from exactly where it left off.

[FAST21] CheckFreq: Frequent, Fine-Grained DNN Checkpointing (from UT Austin & MSR)

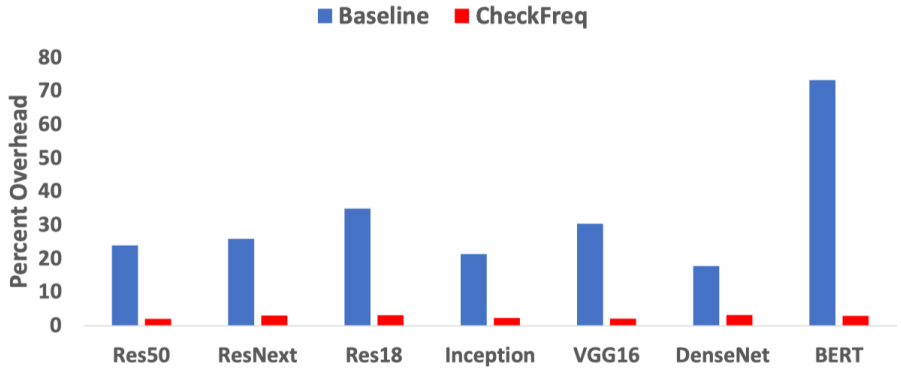
CheckFreq: fine-grained and pipelined checkpointing

Evaluation on 7 different DNNs:

1. CheckFreq reduces checkpoint stalls



2. CheckFreq reduces checkpoint overhead

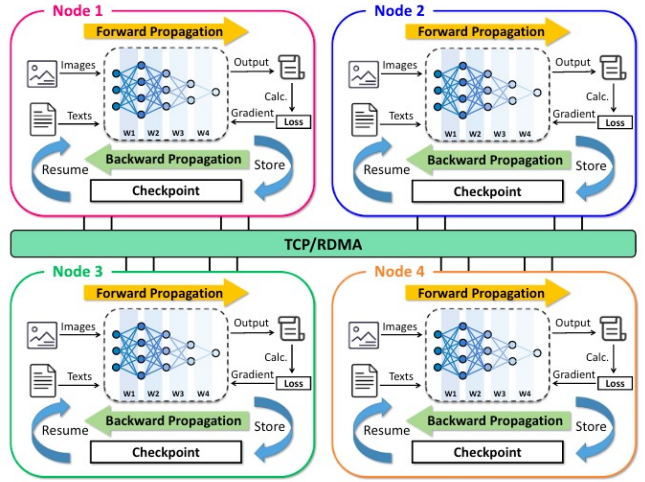


3. CheckFreq lowers recovery time: from hours to seconds

Model	Epoch-based (s)	CheckFreq (s)
Res18	840	5
Res50	2100	24
VGG16	5700	25
ResNext	7080	32
DenseNet	2340	7
Inception	3000	27
BERT	4920	85

[FAST21] CheckFreq: Frequent, Fine-Grained DNN Checkpointing (from UT Austin & MSR)

LightCheck: pipelined checkpoint to PM



- Existing methods (e.g., CheckFreq) still cannot fully utilize the parallelism among computation, communication, and checkpointing

LightCheck pipelines checkpointing with comp. and comm. in a layer-wise way

- Leverage the data dependency.
- Multiple nodes synchronize the model parameters layer-by-layer during communication.

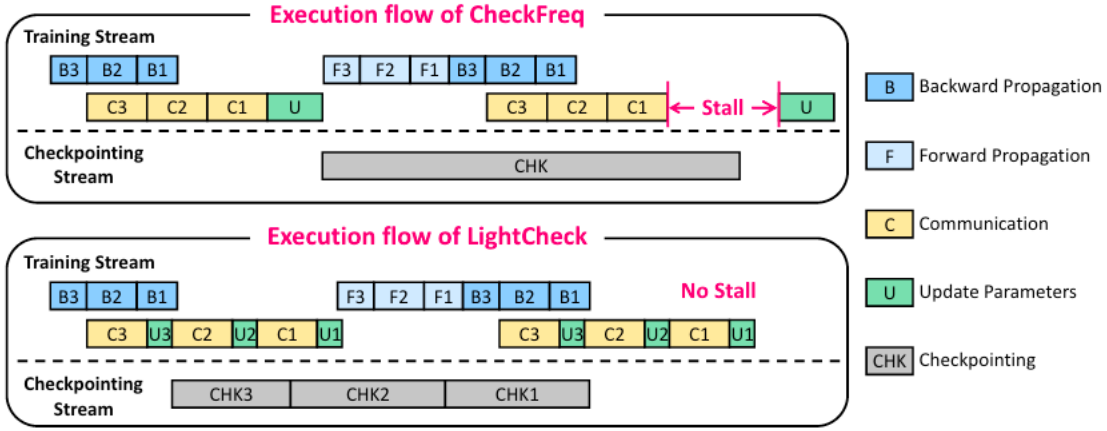


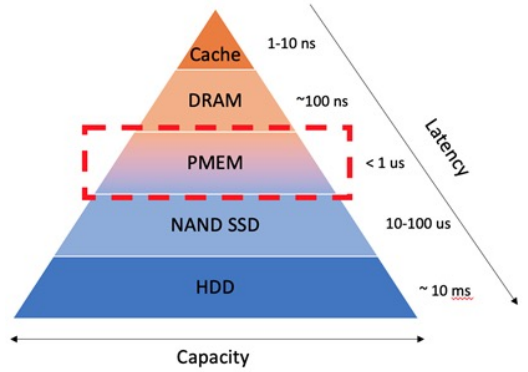
Fig. 3: The execution flows of LightCheck and CheckFreq.

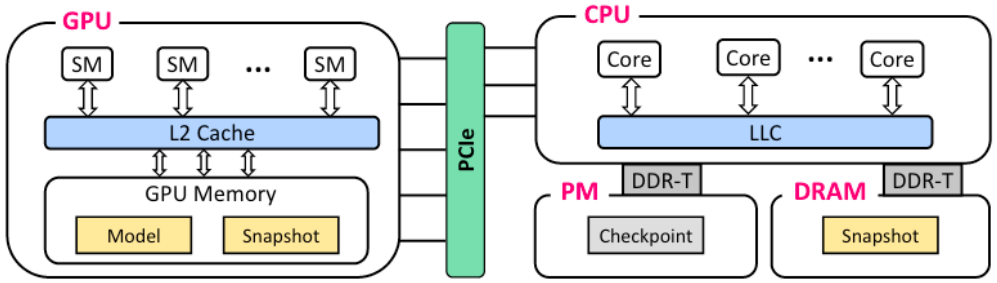
[ICCD23] A Cost-Efficient Failure-Tolerant Scheme for Distributed DNN Training (from HUST)

LightCheck: pipelined checkpoint to PM

Persistent Memory (PM) has received extensive attention

- Large capacity with near-DRAM performance
- The unified virtual addressing (UVA) technique enables zero-copy access over PCIe between GPU and PM
- The asynchronous layer-wise checkpointing is done via CUDA streams and events





Evaluation

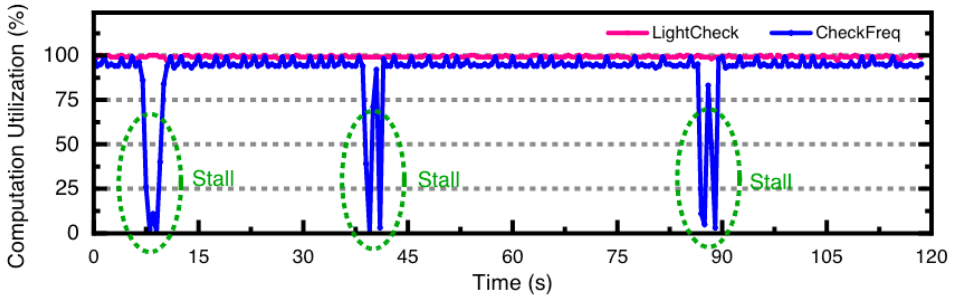


Fig. 8: The GPU computation utilizations of LightCheck and CheckFreq.

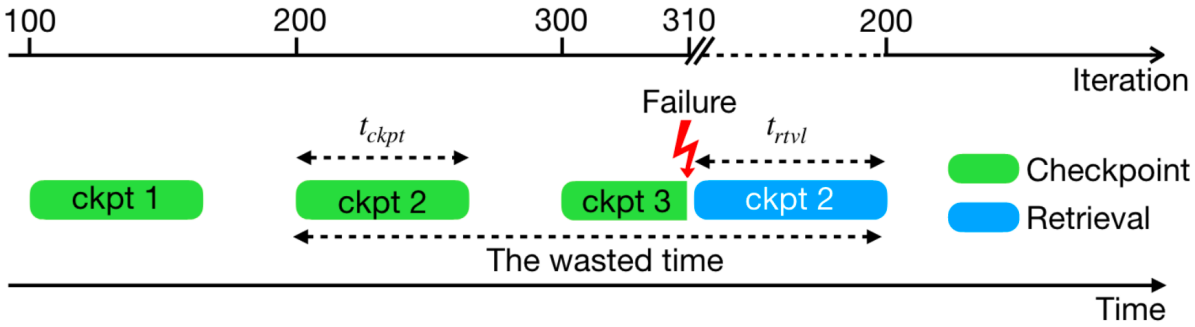
Models	Total Training Time (h)			
	No Failure	*LightCheck*	*CheckFreq*	*torch.save*
ResNet-18	10.7	10.9	11.1	11.2
VGG_16	67.5	69.7	72.3	73.7
Inception-V3	79.7	83.0	84.1	85.4
AlexNet	6.0	6.1	6.3	6.5
GPT-2	161.7	164.6	171.1	179.7
BERT	501.3	514.8	537.5	598.6

[ICCD23] A Cost-Efficient Failure-Tolerant Scheme for Distributed DNN Training (from HUST)

GEMINI: hierarchical storage of checkpoints

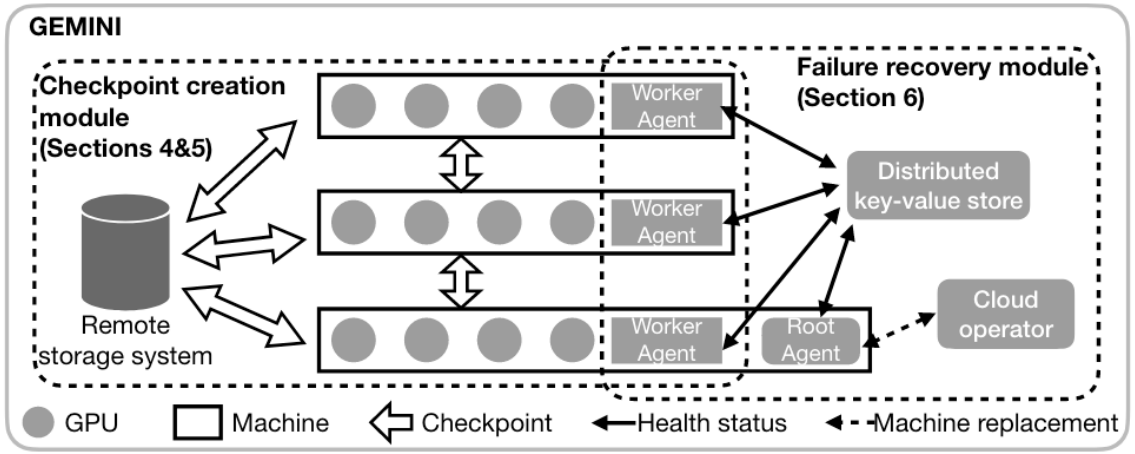
Three main factors in checkpoint:

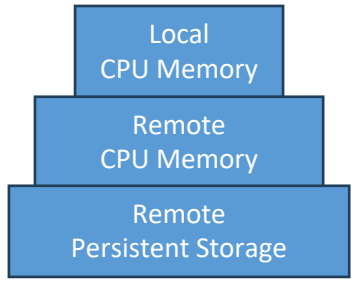
- checkpoint time
- checkpoint frequency
- retrieval time



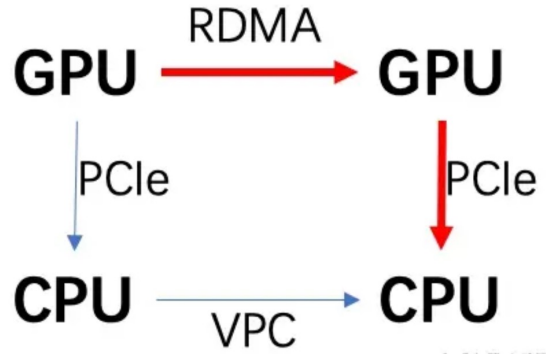
Leverage the high bandwidth of CPU memory to achieve fast failure recovery!

- How to maximize the probability of a successful failure recovery from CPU memory?
- how to minimize the interference of checkpoint traffic with model training?





hierarchical storage

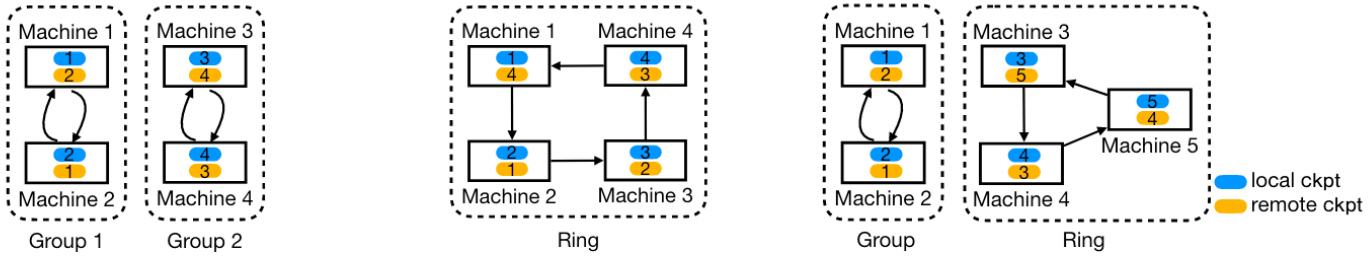


communications

[SOSP23] Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints (from Rice & Amazon)

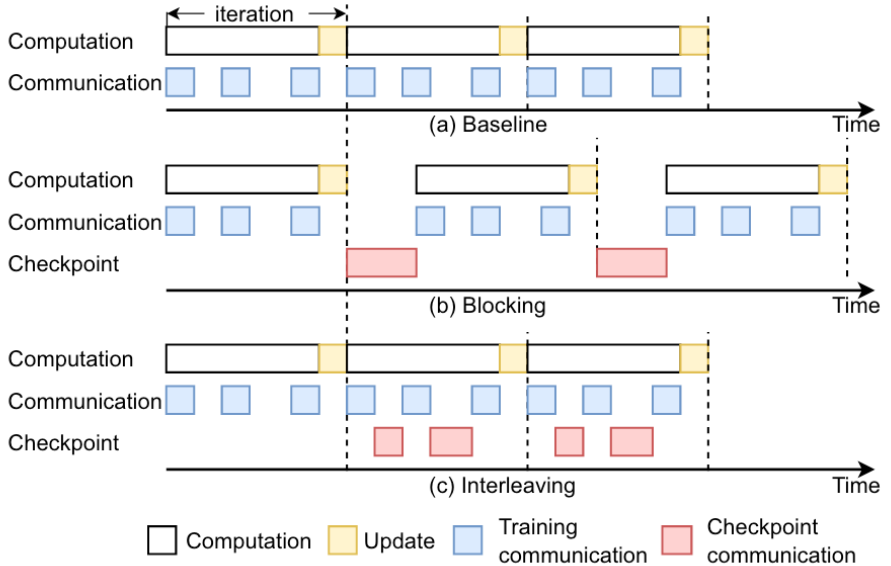
GEMINI: hierarchical storage of checkpoints

- Maximize the probability of a successful failure recovery from CPU memory
  - **Store redundant checkpoints and proposes a placement strategy that maximizes the probability**



(a) Group placement strategy. **(b)** Ring placement strategy. **(c)** Mixed placement strategy.

- Minimize the interference of checkpoint traffic with model training?
  - **Design a deliberate communication scheduling algorithm for interleaving these two types of traffic to minimize the interference on training throughput.**
  - Method: online profiling for several iterations of training (e.g., 20) without checkpointing.



[SOSP23] Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints (from Rice & Amazon)

GEMINI: hierarchical storage of checkpoints

- Other difficulties and approaches

Difficulty

Extra GPU memory consumption

Approach

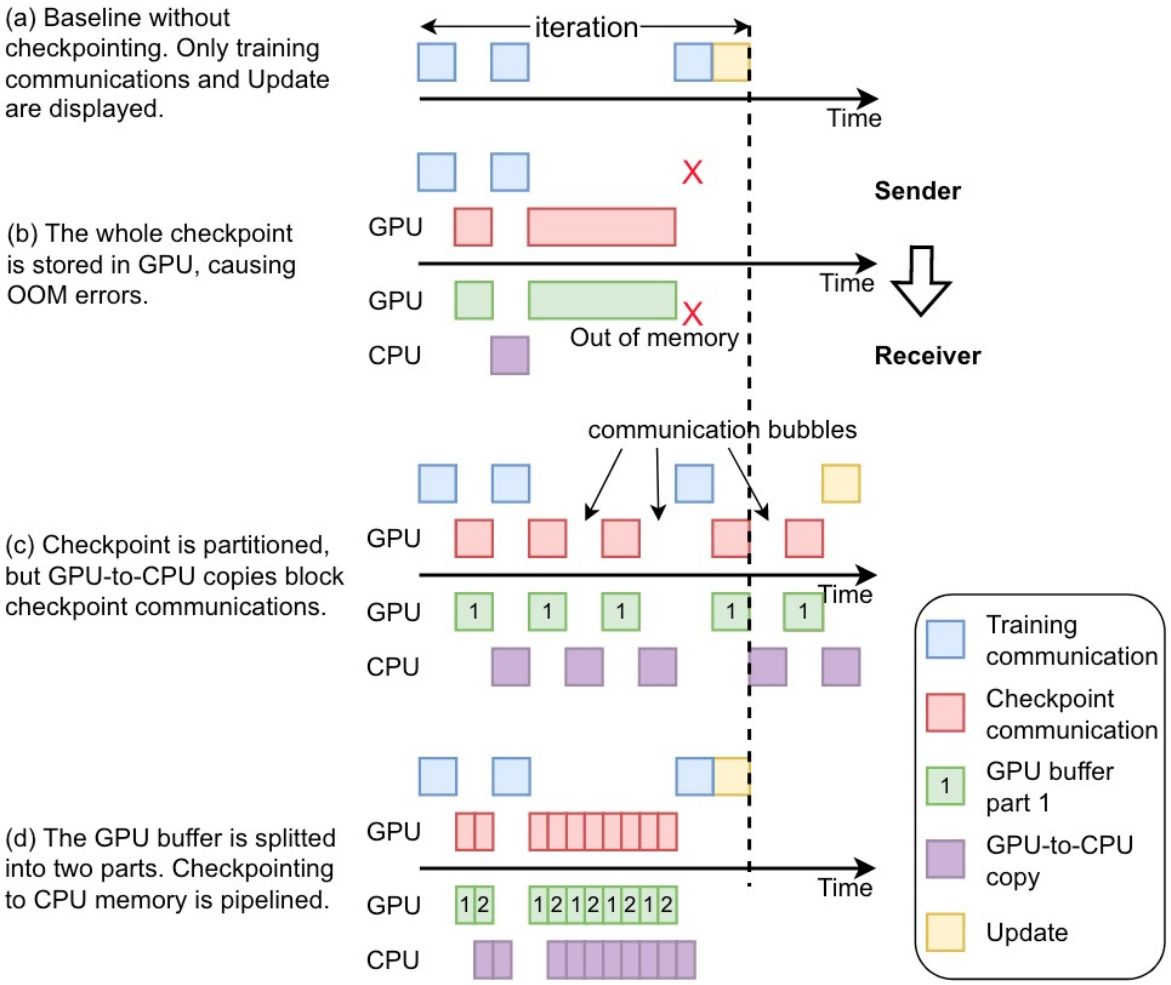
Partitioning checkpoint

Difficulty

Local GPU-to-CPU copy overhead.

Approach

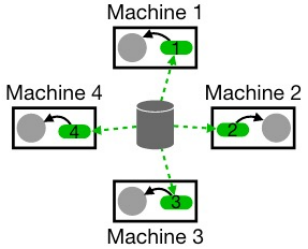
Pipelining checkpoint transmission.



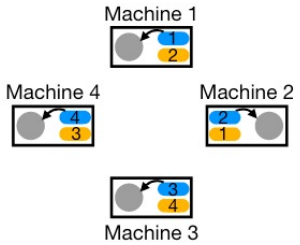
[SOSP23] Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints (from Rice & Amazon)

GEMINI: hierarchical storage of checkpoints

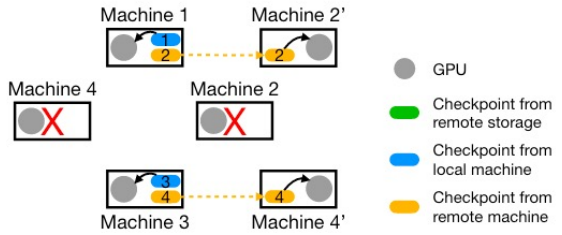
- Recovery training from failures
  - software failures
  - hardware failures
    - replicas exist
    - replicas not exist



(a) Existing solutions for any types of failures. All checkpoints are always retrieved from the remote persistent storage.



(b) GEMINI for software failures. Checkpoints are at local and the retrieval time is negligible.



(c) GEMINI for failures with two machines replaced. The newly added machines retrieve checkpoints from alive machines.

- Evaluation

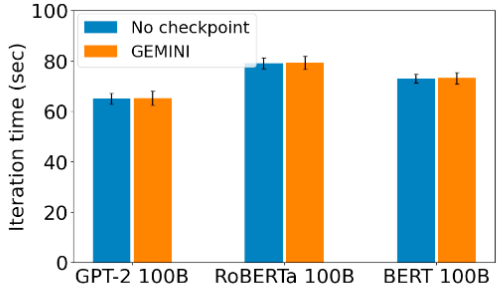


Figure 7. The iteration time of three large models without checkpoints and with GEMINI.

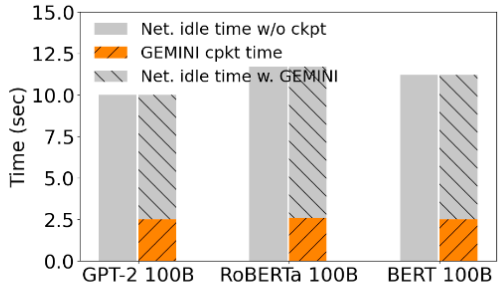
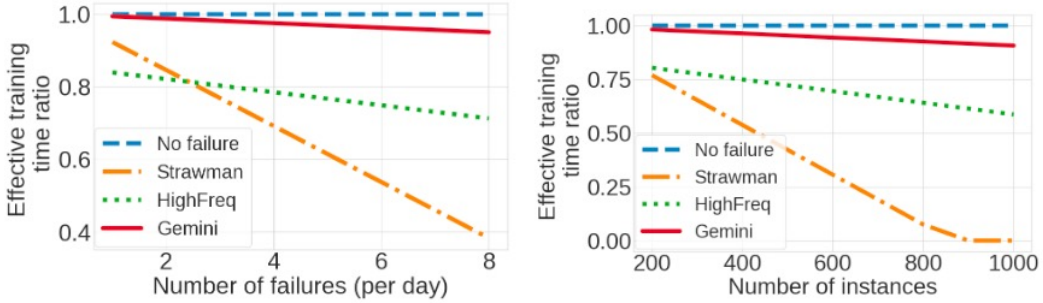
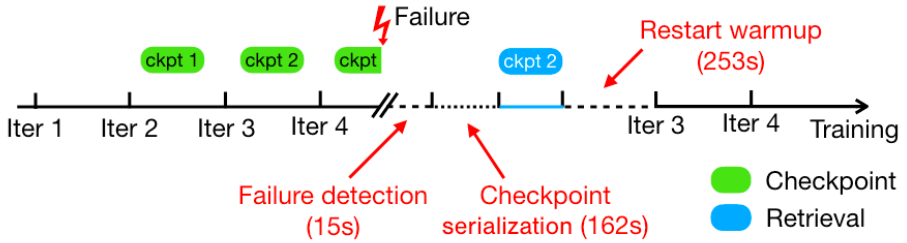


Figure 8. The network idle time of three large models without checkpoints and with GEMINI.



(a) Different failure rates. (b) Different instance numbers.



[SOSP23] Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints (from Rice & Amazon)

SWIFT: update-undo and logging-based recovery

- The overhead of checkpointing is high
- **Crash-consistency problem**: parameters from some workers are updated while the others are not.

Swift Design:

- Update-undo (replica available):
  - survivors undo the update for the updated parameters
- Logging-based recovery (replica unavailable):
  - is done asynchronously
  - records intermediate communication (i.e., intermediate activations in the forward pass and the gradients in the backward pass)
  - replacement workers download logs for recovery

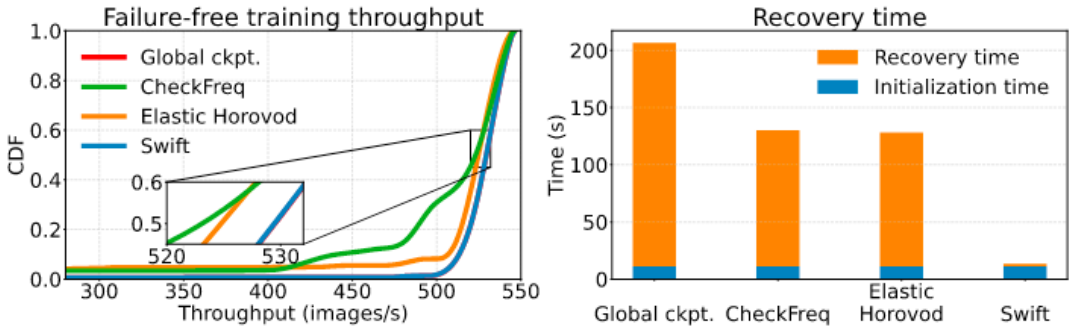


Figure 1. Replication-based recovery for Wide-ResNet-50.

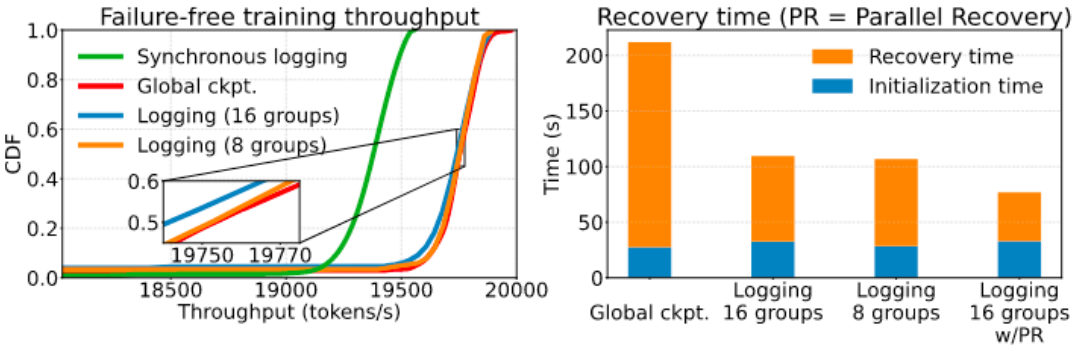


Figure 2. Logging-based recovery for BERT-128.

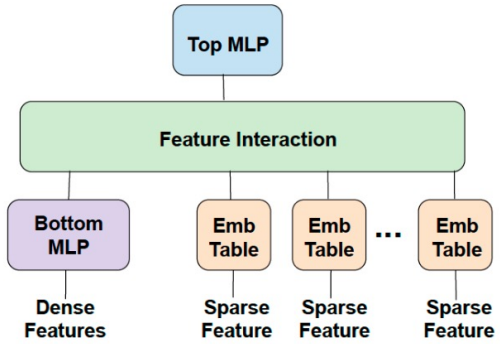
[PPoPP23] Swift: Expedited Failure Recovery for Large-scale DNN Training (from HKU & OneFlow)

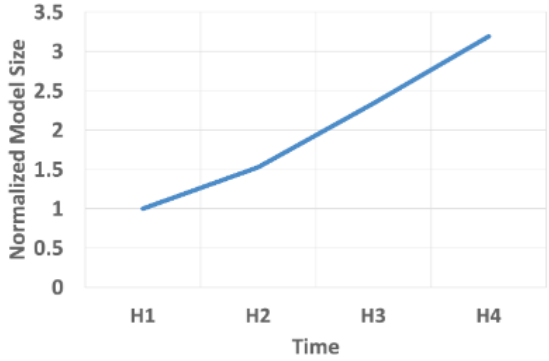
Check-N-Run: reduce checkpoint size

- Checkpoints of large-scale recommendation system (RS) is essential

Failure recovery | Migrating training jobs

Transfer learning | Publishing snapshots



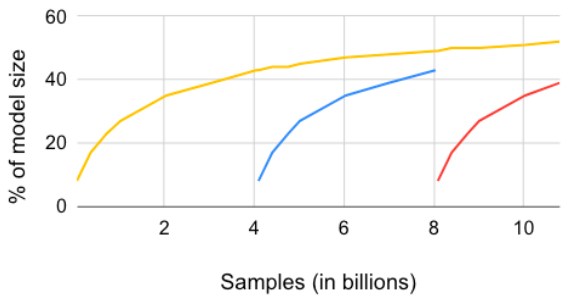


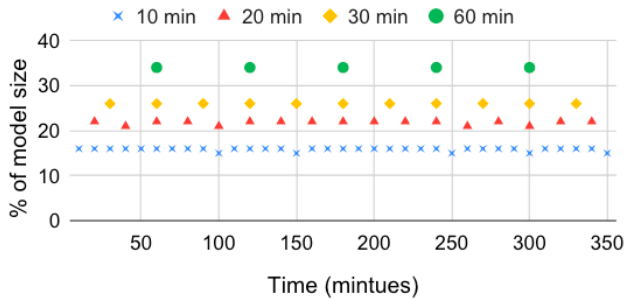
Recommendation Model Architecture

- Challenges of checkpointing

Accuracy | Write bandwidth

Frequency | Storage capacity



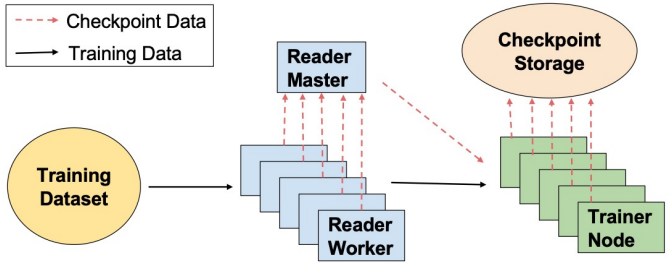


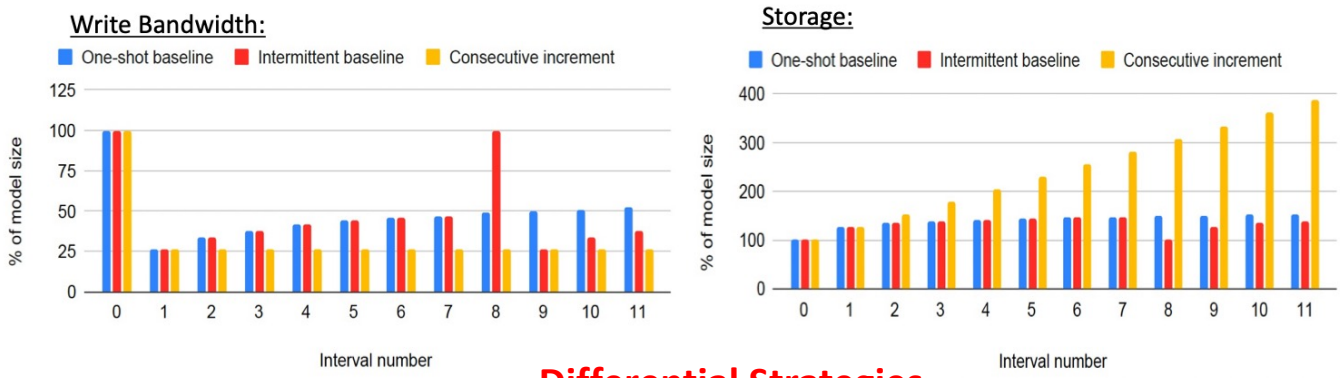
at each iteration only a tiny fraction of the model is updated

[NSDI22] Check-N-Run: a Checkpointing System for Training Deep Learning Recommendation Models (from Facebook)

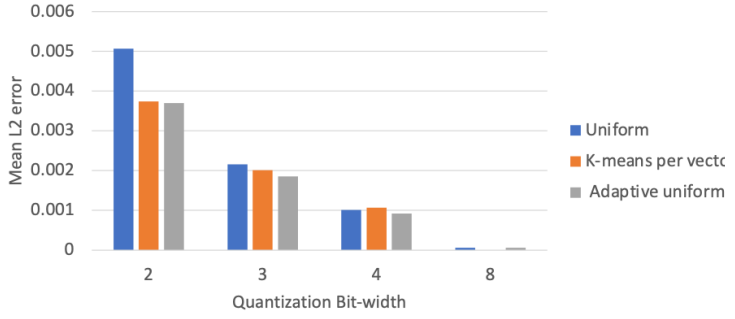
Check-N-Run: reduce checkpoint size

- Strategy 1: differential checkpointing
  - Motivation: model accesses are sparse
  - One-Shot Differential Checkpoint
  - Consecutive Incremental Checkpoint
  - Intermittent Differential Checkpoint
- Strategy 2: checkpoint quantization
  - Compress checkpoint without degrading training accuracy
  - Uniform quantization
  - Non-uniform quantization using k-means
  - Adaptive uniform quantization
- Strategy 3: Decoupling
  - Separate snapshot and persist operations.

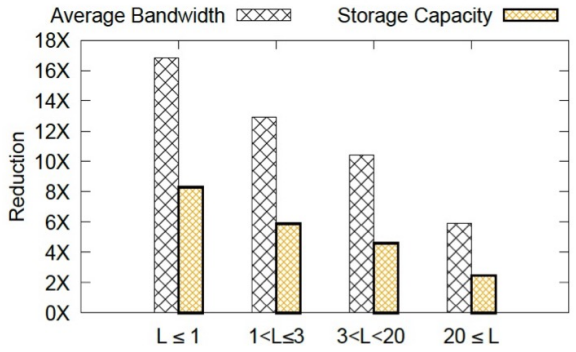




Differential Strategies



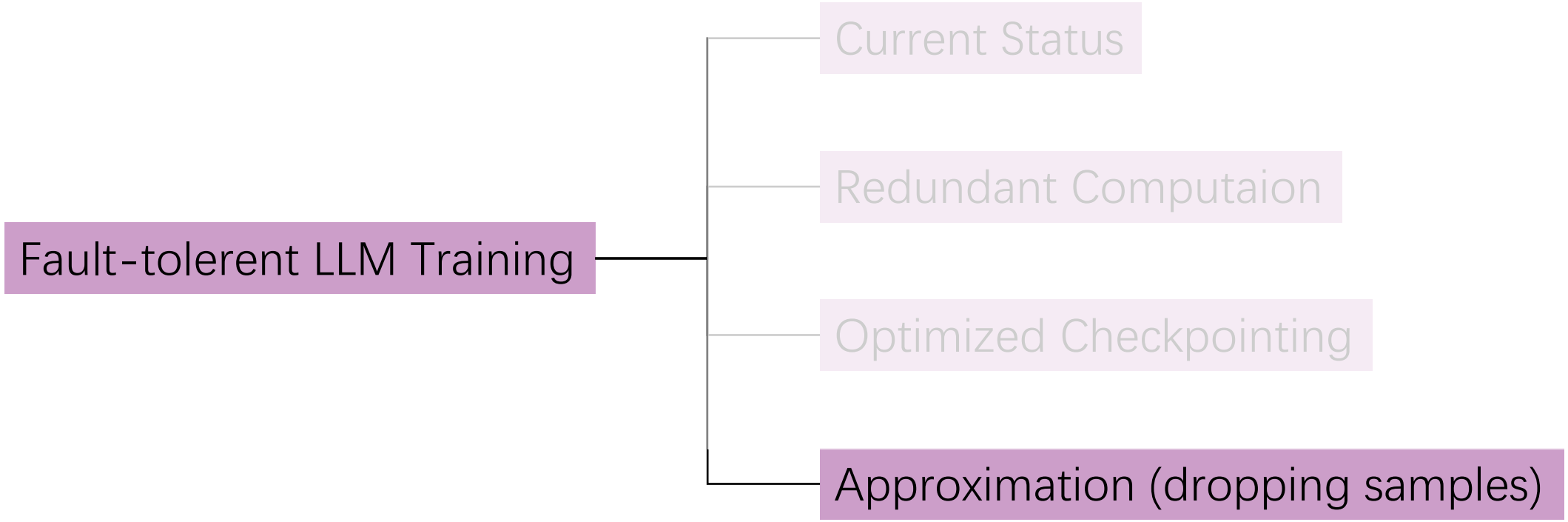
Quantization Strategies



Overall Results

[NSDI22] Check-N-Run: a Checkpointing System for Training Deep Learning Recommendation Models (from Facebook)

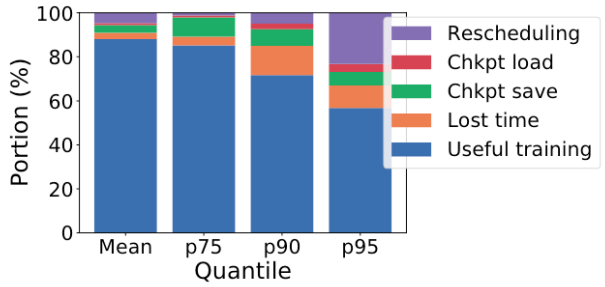
Outline


Fault-tolerent LLM Training
Current Status
Redundant Computaion
Optimized Checkpointing
Approximation (dropping samples)

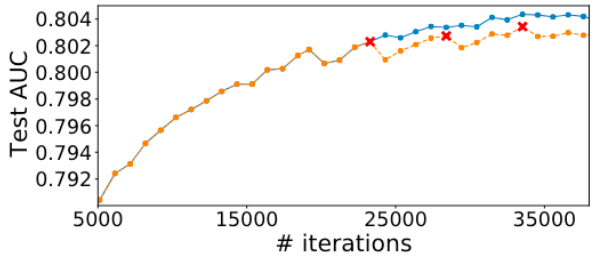
CPR: trade-off between overhead and accuracy

Failures are frequent in large-scacle model training

- **Full recovery** will lead to the lost of computation
- **Partial recovery** can harm the model accuracy



Overhead of full recovery



Accuracy of partial recovery

Can we balance between the overhead and the accuracy?

Portion of Lost Samples (PLS)

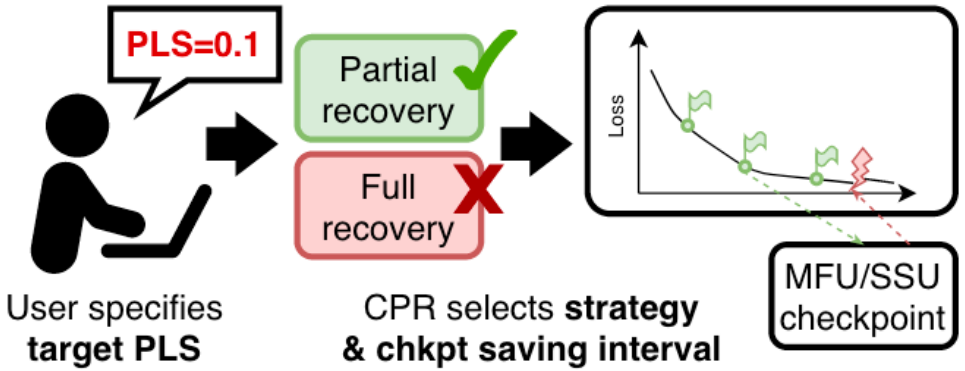
$$\mathbb{E}[PLS] = \frac{0.5 T_{save}}{T_{fail} N_{emb}}$$

- the portion of the training data samples whose effect on the model was lost due to a failure
- is a function of *checkpoint saving interval*, *the failure rate* and *the number of parameter server nodes*
- can be used as a metrics to trade-off the **performance overhead** and **the model accuracy**

CPR: Understanding and Improving Failure Tolerant Training for Deep Learning Recommendation with Partial Recovery (from Facebook)

CPR: trade-off between overhead and accuracy

CPR selects between full recovery and partial recovery based on the benefit analysis

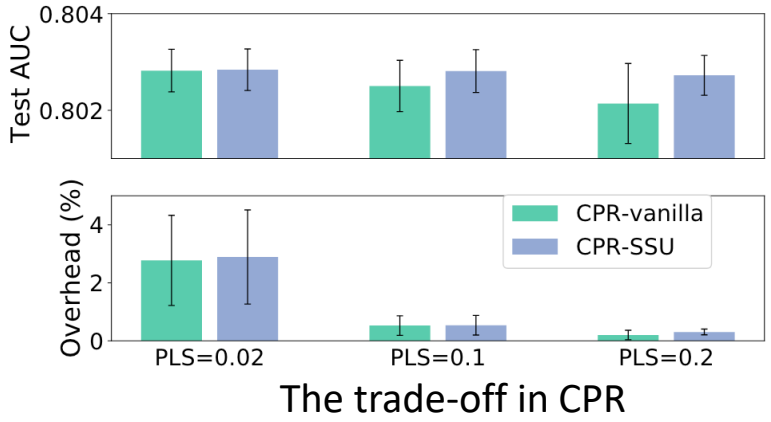


- PLS-based checkpointing
  - first choose the saving frequency based on specified PLS
  - trade-off between the overheads of two strategies
    - if the expected benefit is small, choose full recovery
    - otherwise, choose partial recovery

$$O_{total} \approx O_{save}\frac{T_{total}}{T_{save}} + (O_{load} + \frac{T_{save}}{2} + O_{res})\frac{T_{total}}{T_{fail}} \quad (1)$$

$$O_{total\_par} \approx O_{save}\frac{T_{total}}{T_{save}} + (O_{load} + O_{res})\frac{T_{total}}{T_{fail}} \quad (2)$$

- Frequency-based prioritization
  - With the limited I/O bandwidth, prioritizing to save important updates can make the final model quality to improve.
  - SCAR: prioritize saving parameters with larger changes
  - CPR-MFU: prioritize saving the most-frequently-used parameters
  - CPR-SSU: sub-sample and prioritize used embedding vectors



The trade-off in CPR

CPR: Understanding and Improving Failure Tolerant Training for Deep Learning Recommendation with Partial Recovery (from Facebook)

Conclusion

Redundant Computation

- Pipeline RC in the bubble time to avoid high overhead
- Leverage the inherent redundancy in DP
- Decoupling the planning from the execution

Optimized Checkpointing

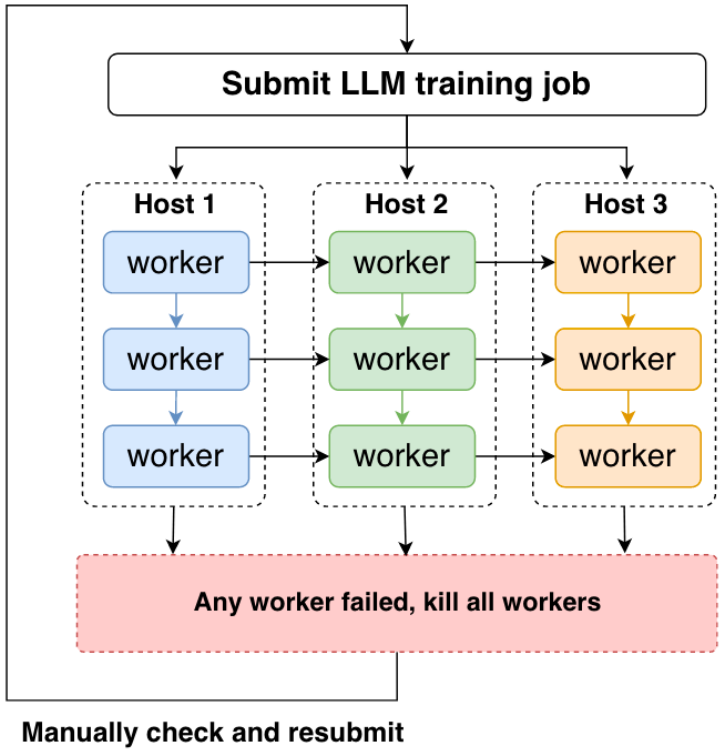
- Pipeline checkpointing with computation and communication.
- Leverage hierarchical storage to improve checkpointing frequency
- Avoid the lost of computation
- Leverage partial features and quantization to reduce size

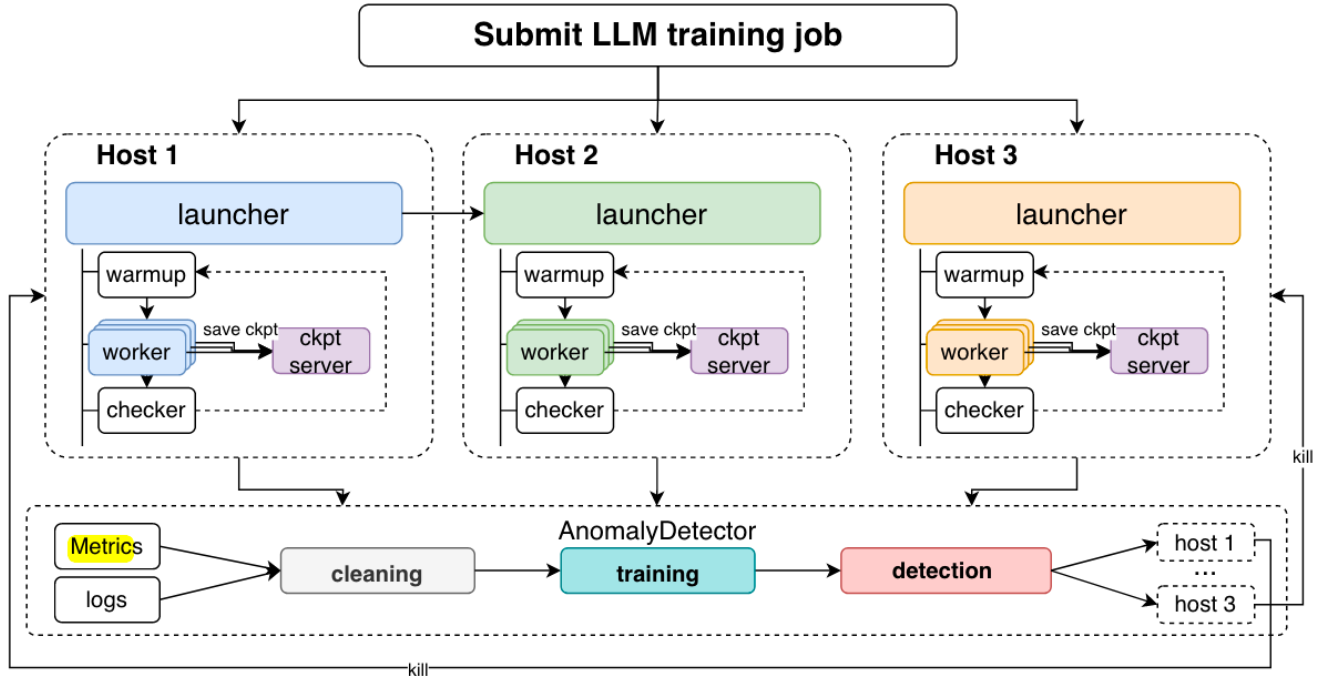
Approximation

- Trade-off between the overhead of full recovery and accuracy of partial recovery

Conclusion

Future direction





Automated anomaly detection

Efficient failure recovery

Thank you!

ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong