ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Greyhound: Hunting Fail-Slows in Hybrid-Parallel Training at Scale

Tianyuan Wu† , Wei Wang† , Yinghao Yu§ , Siran Yang§ , Wenchao Wu§ , Qinkai Duan† , Guodong Yang§ , Jiamang Wang§ , Lin Qu§ , Liping Zhang§

†Hong Kong University of Science and Technology §Alibaba Group

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

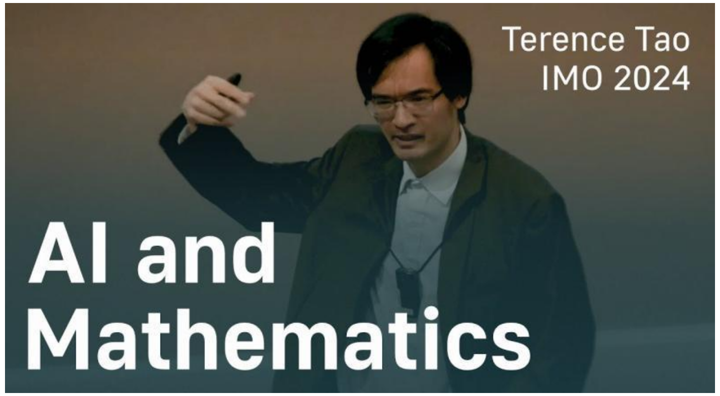
Alibaba Group 阿里巴巴集团

Agenda

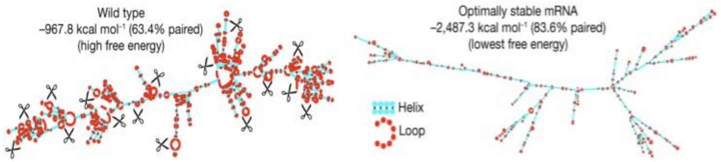
- Reliability issues in large-scale training.
- How do stragglers manifest in hybrid-parallel training at scale?
- How can stragglers be detected rapidly?
- How should stragglers be mitigated effectively?
- How do our detection and mitigation solutions perform?

The Rapid Scaling of Models and Clusters

AI-powered forecasting
Huawei PanGu LLM **200B**



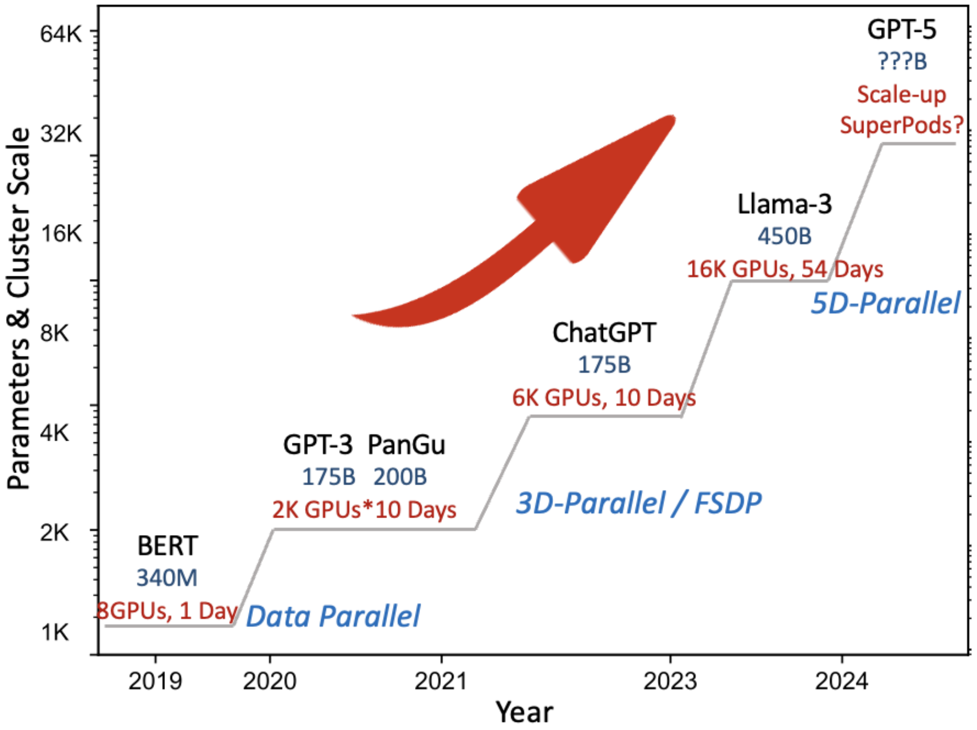
AI for mathematics
GPT-4 **1000B?**



Baidu's AI-assisted mRNA design optimization featured in Nature
HelixFold, ERNIE **260B**

The Grand Breakthrough of Large Models

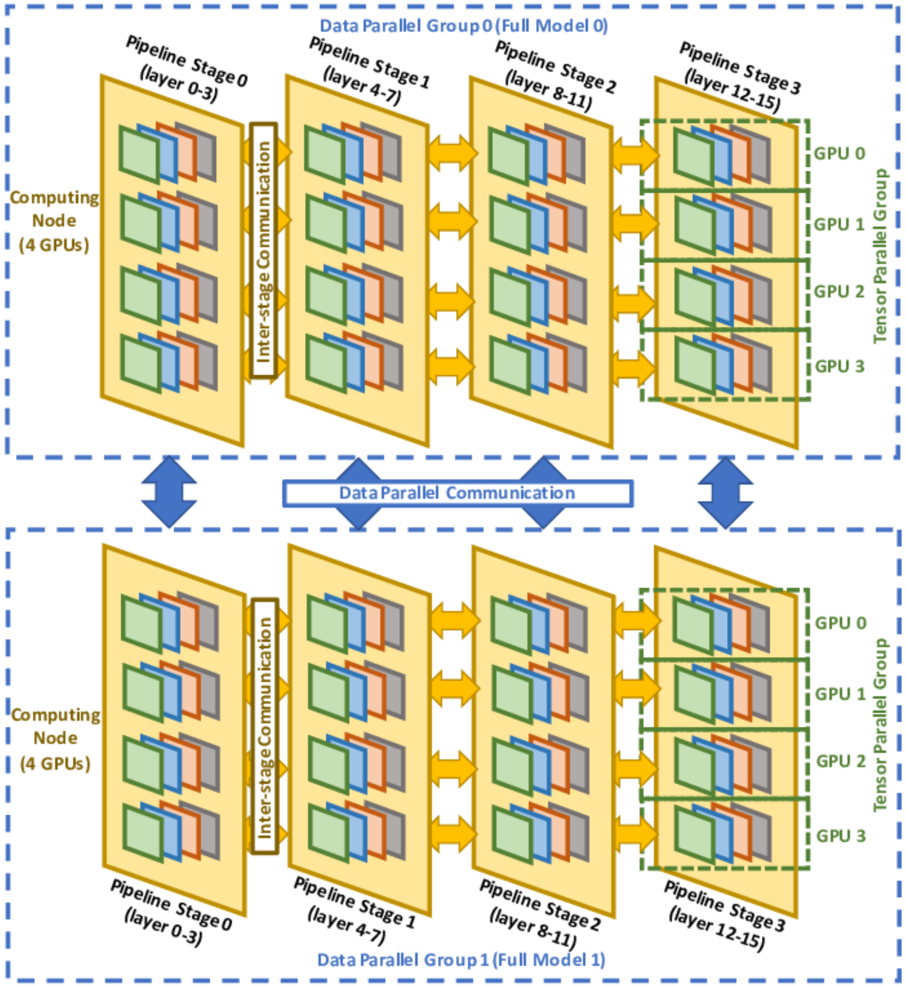
LLM model sizes scale 8x every two years



- Model sizes grow 30,000x from 2019 to 2025
- Training scales from 8 to 100k GPUs since 2019
- Parallel strategies are evolving rapidly

Distributed Large Model Training at Scale

- **Tensor parallelism (TP):** partition individual layers of a model over multiple devices
- **Data parallelism (DP):** shard training dataset and replicate the model
- **Pipeline parallelism (PP):** partition a model into layer groups, each being a pipeline stage
- Other specialized parallelism
    - Context parallelism (CP), expert parallelism (EP)
- **Hybrid parallelism:** combine DP, PP, TP, and potentially other parallelisms



Credit: Song et al. “Optimus-CC: Efficient Large NLP Model Training with 3D Parallelism Aware Communication Compression,” in ASPLOS 2023

At hyperscale, failures become the norm, rather than the exception!

Fail-Stop Failures

Complete halt of training due to fatal software/hardware errors

- OPT-175B: 110 errors in two-month training on 1,000 A100 GPUs [1]
- Llama-3: 419 unexpected failures in 54-day training on 16,000 H100 GPUs [2]

Extensively studied over the years

- Restart on checkpoints: CheckFreq (FAST'21), Check-N-Run (NSDI'22), Gemini (SOSP'23)
- Redundant computation & dynamic parallelism adjustments: Bamboo (NSDI'23), Oobleck (SOSP'23), Recycle (SOSP'24)

Category	Source Component	Root Cause
Infrastructure	GPU Processor	Faulty GPU
	GPU Memory	GPU Memory Error
	NIC / Switch	Network / Connection Error
	Host CPU / Mem.	Faulty Host Node
	Disk / Filesystem	Storage I/O Error
	Power Supply	Faulty Power Supply
	Low-level Libraries	MPI / NCCL / CUDA Error
Framework	DLT Framework	Input / Assertion Error
	DLT Framework	Model Checkpoint Error
	Scheduler	Job Preempted
	Dataset Libraries	Dataloader Error
	DLT Framework	CPU/GPU Out of Memory
User Program	Model Architecture	Model Diverged
	Launch Script	OS / Permission Error
	Launch Script	Invalid Mem. Access / SegFault
	Launch Script	Import Error

[1] Zhang et al., "Opt: Open pre-trained transformer language models," in arXiv:2205.01068, 2022.
[2] Grattafiori, Aaron, et al. "The llama 3 herd of models," in *arXiv:2407.21783*, 2024.

Fail-Slow Failures (Stragglers)

Components still functioning but slow

- Degraded computation: slow CPUs and GPUs
- Degraded communication: network/link congestion

“Sometimes, hardware issues may cause ***still-functioning but slow stragglers*** that are ***hard to detect***. ***Even a single straggler can slow down thousands of other GPUs***, often appearing as functioning but slow communications.”

Meta

Despite their prevalence, straggler problems remain not well studied

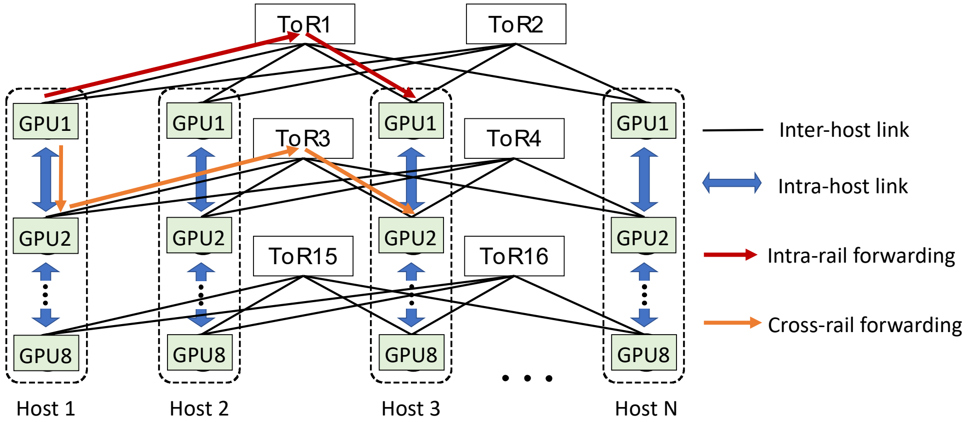
Agenda

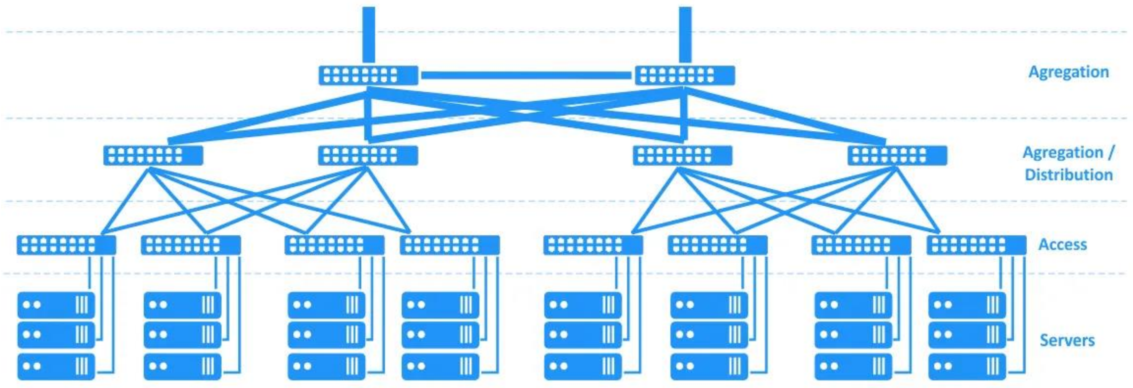
- Reliability issues in large-scale training.
- How do stragglers manifest in hybrid-parallel training at scale?
- How can stragglers be detected rapidly?
- How should stragglers be mitigated effectively?
- How do our detection and mitigation solutions perform?

Straggler Characterization: Cluster Setup

Alibaba's HPAI *multi-tenant* cluster for training & inference

- **10,000** GPUs: 1,800x H800, 2,600x A100, 5000+ Other GPUs
- **RoCEv2 Network:** 4x 400 Gbps NICs for H800, 4x 200 Gbps NICs for A100 node
- **Workloads**: LLM training (majority), recommendation training, LLM inference
- **Scheduler**: Customized K8S scheduler





Credit: Qian et al., "Alibaba HPN: A Data Center Network for Large Language Model Training," in ACM SIGOMM 2024

Straggler Characterization: Methodology

Cluster sampling

- Repeatedly submit a large number of small **probing jobs**, which are *randomly* scheduled
- **Probing jobs:** specially designed to detect slow computation and/or communication
    - Type-A for slow computation: 4x H800 on 1 node, GPT-2 11B, 2TP-2PP, 10K iterations
    - Type-B for slow communication: 8x A100 on 4 noes, GPT-2 7B, 2TP-4DP, 10K iterations
- Sampling coverage:
    - 400x Type-A jobs covering **500/1,800** H800
    - 107x Type-B jobs covering **690/2,600** A100

Manual inspection of training log traces

- Collected log traces of large training jobs in one month, from July 1 to 31, 2024
- 27 Jobs in total, each requiring **>=512 GPUs**

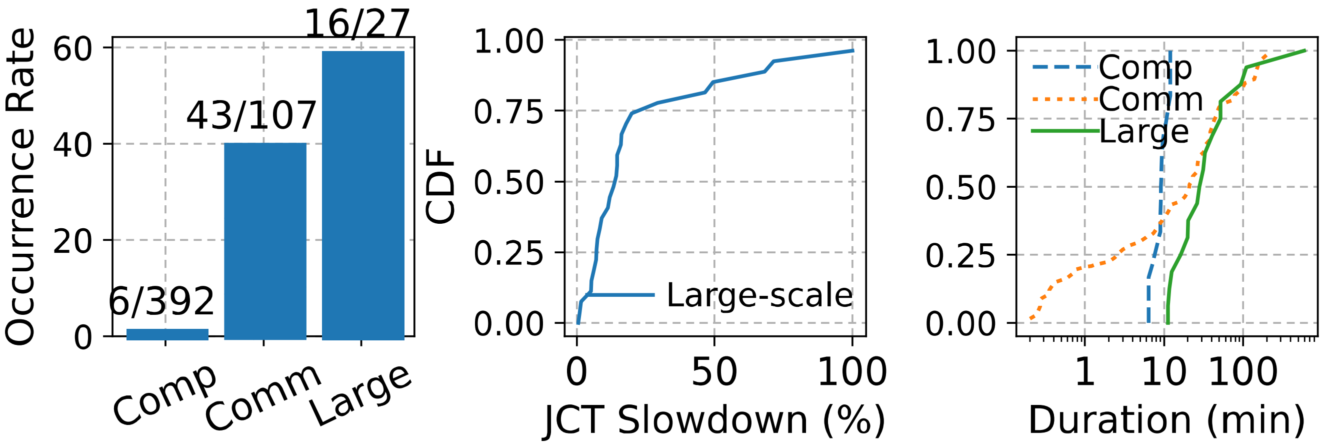
Straggler Characterization: Overview

Cluster sampling

- Computation stragglers: **less frequent, low impact**
- Communication stragglers: **frequent, high impact**

Trace inspection for LLM training

- Mean straggler duration: **72 mins**
- Avg training slowdown: **34.59%**

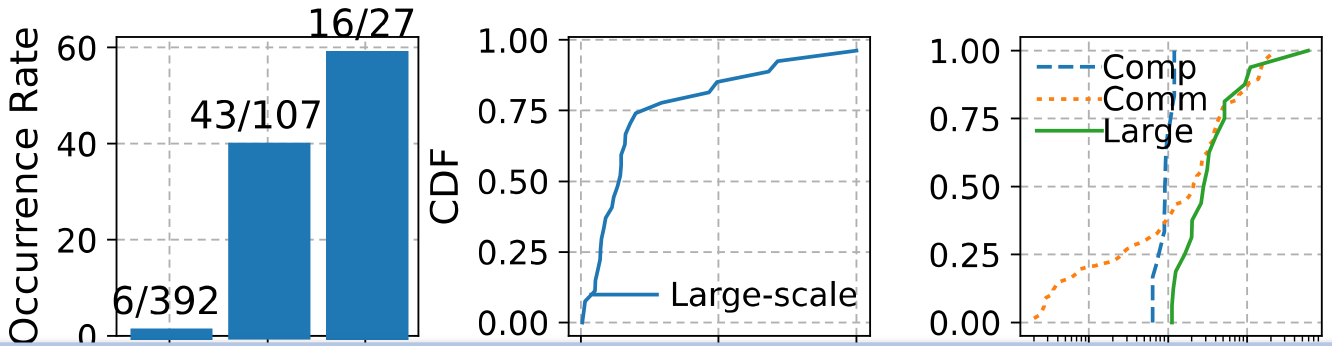


Category	Online Probing		Off ine Inspection
	1-Node	4-Node	At Scale (≳512 GPUs)
No fail-slow	386	64	11
CPU Contention	4	1	0
GPU Degradation	2	0	0
Network Congestion	0	42	13
Multiple Issues	0	0	3
Total # Jobs	392	107	27
Avg. JCT Slowdown	11.79%	15.45%	34.59%

Straggler Characterization: Overview

Cluster sampling

- Computation stragglers: **less frequent, low impact**
- Communication stragglers: **frequent, high impact**



Stragglers are transient, frequent, and can cause significant slowdown!

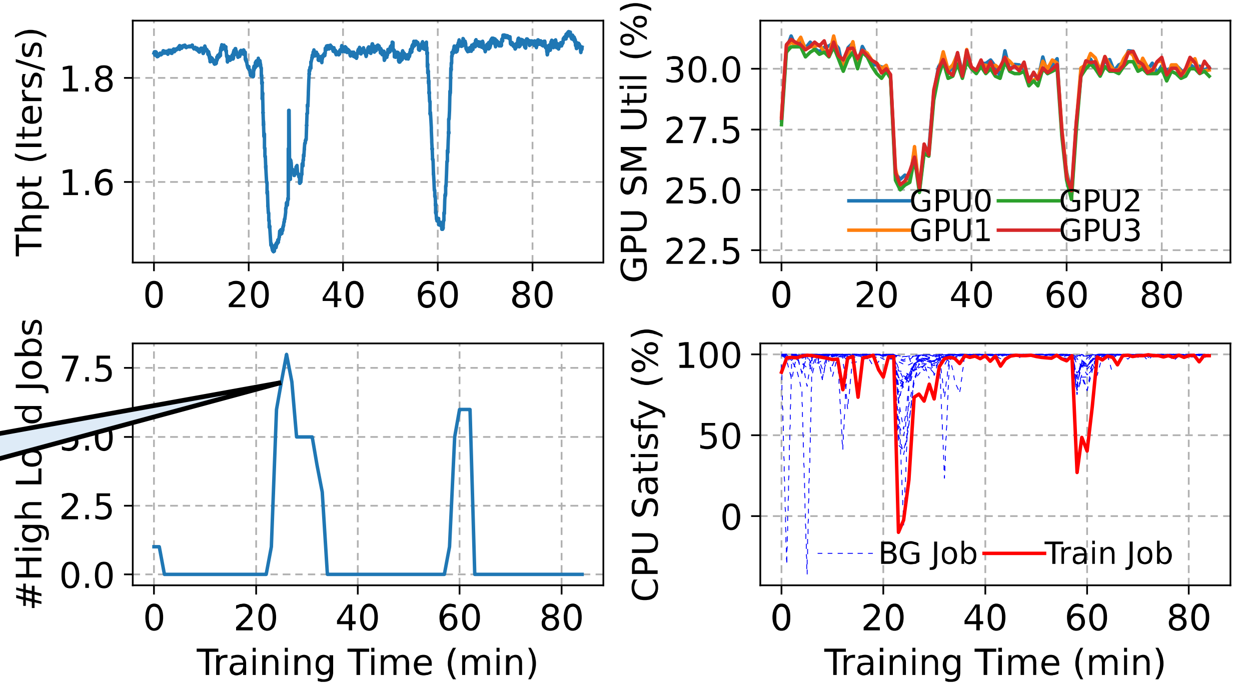
- Avg training slowdown: **34.59%**

Category			
	1-Node	4-Node	At Scale ($\gtrsim$512 GPUs)
No fail-slow	386	64	11
CPU Contention	4	1	0
GPU Degradation	2	0	0
Network Congestion	0	42	13
Multiple Issues	0	0	3
Total # Jobs	392	107	27
Avg. JCT Slowdown	11.79%	15.45%	34.59%

Computation Stragglers: CPU Contention

- Multiple collocated jobs contend for host CPUs
- Occasional occurrence
  - ~1%, 4/392 jobs
- Short-lived
  - mean duration: ~10 mins

CPU burst of BG jobs → CPU contention
→ More time spent on CPU operations
→ training slowdown



A case of CPU contention

Computation Stragglers: GPU Degradation

- Mainly due to **thermal throttling**
  - High temperature, e.g., >70°C
- Occasional occurrence
  - ~0.5%, 2/392 jobs
- Short-lived
  - ~10 mins mean duration

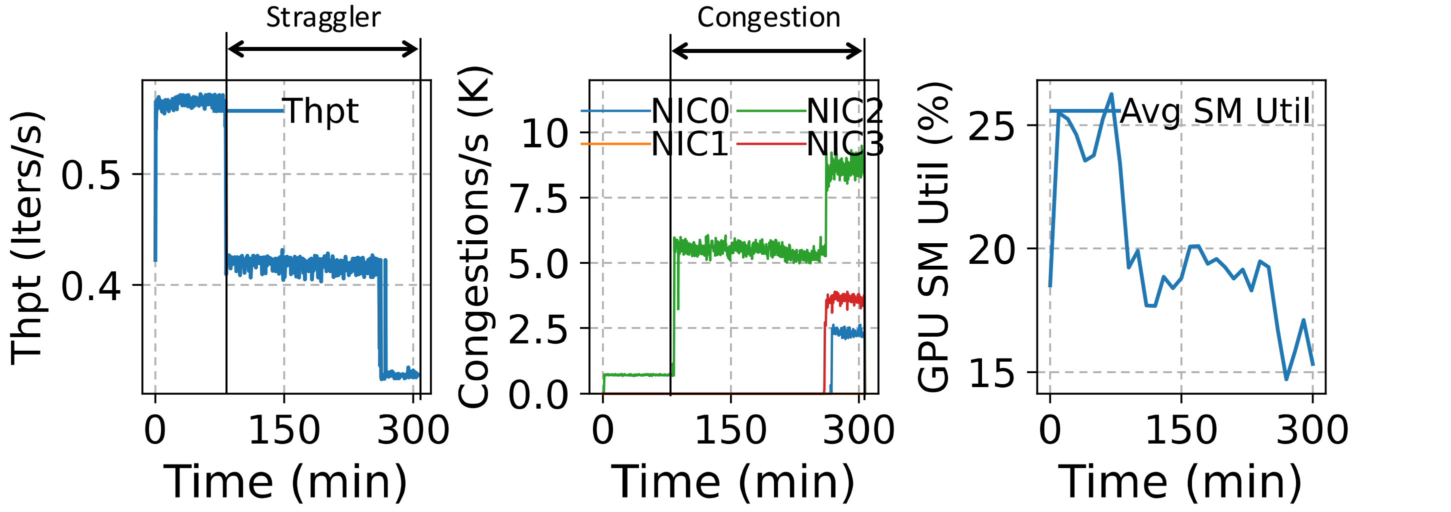
GPU0 measured high temperature, resulting in thermal throttling

A case of GPU degradation

Communication Stragglers: Congestion

Network congestion

- High NP_CNP_SENT/MARK/HANDLED recorded during fail-slow
- High occurrence frequency: **~40%** of 4-node jobs (42/107)
- Long duration: **~24** mins



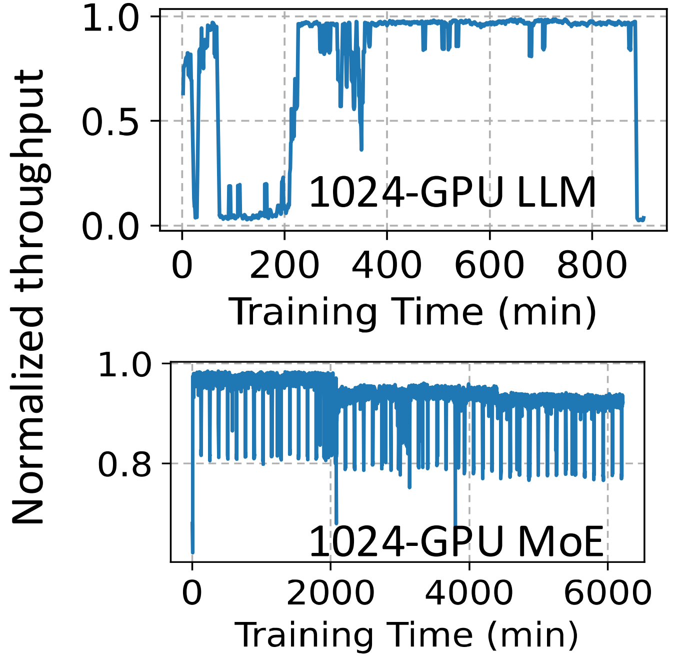
A case of network congestion

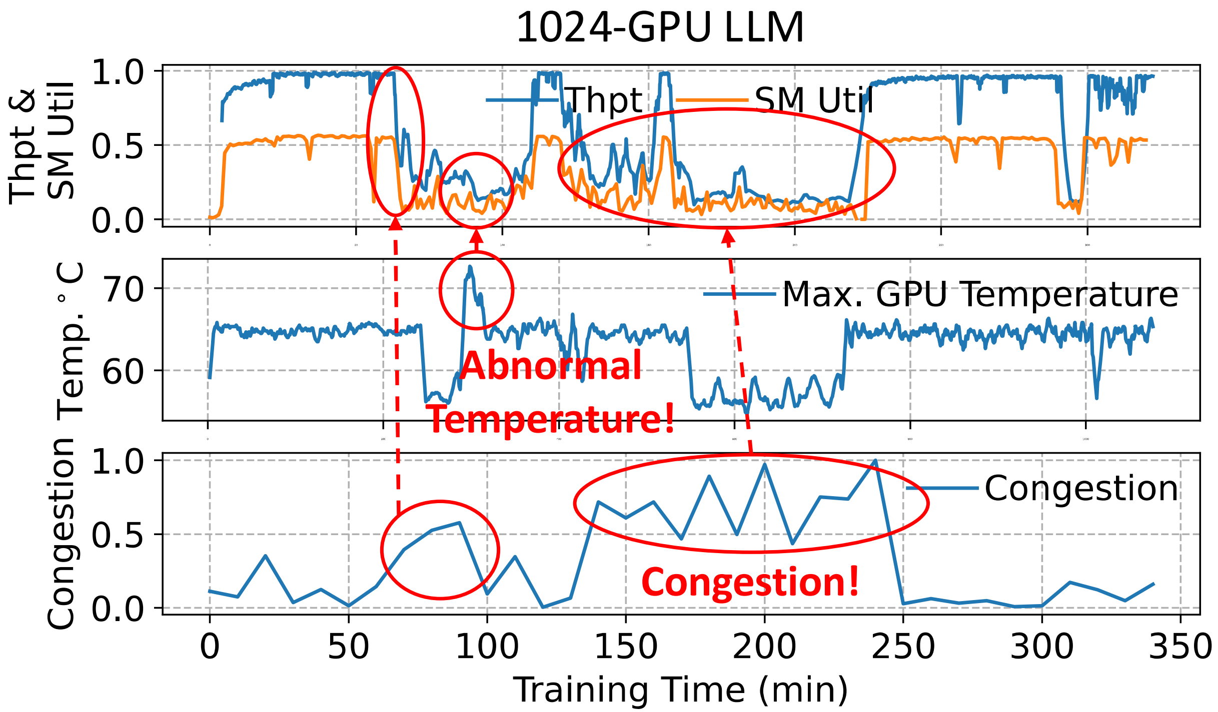
Intra-node interconnects are stable,
Inter-node RDMA has large variance

Comm. Type	Intra-Node		Inter-Node
	NVL	PIX	RDMA
CoV	0.02	0.09	**0.29**

Stragglers at Scale: Trace Analysis

- **16/27 (~60%)** training jobs experienced stragglers, mean duration **~72 mins**
  - Measured up to **90%** throughput loss in 1024-GPU jobs
  - Computation and communication stragglers may occur *simultaneously*
  - Performance across iterations can vary significantly





Three Takeaways

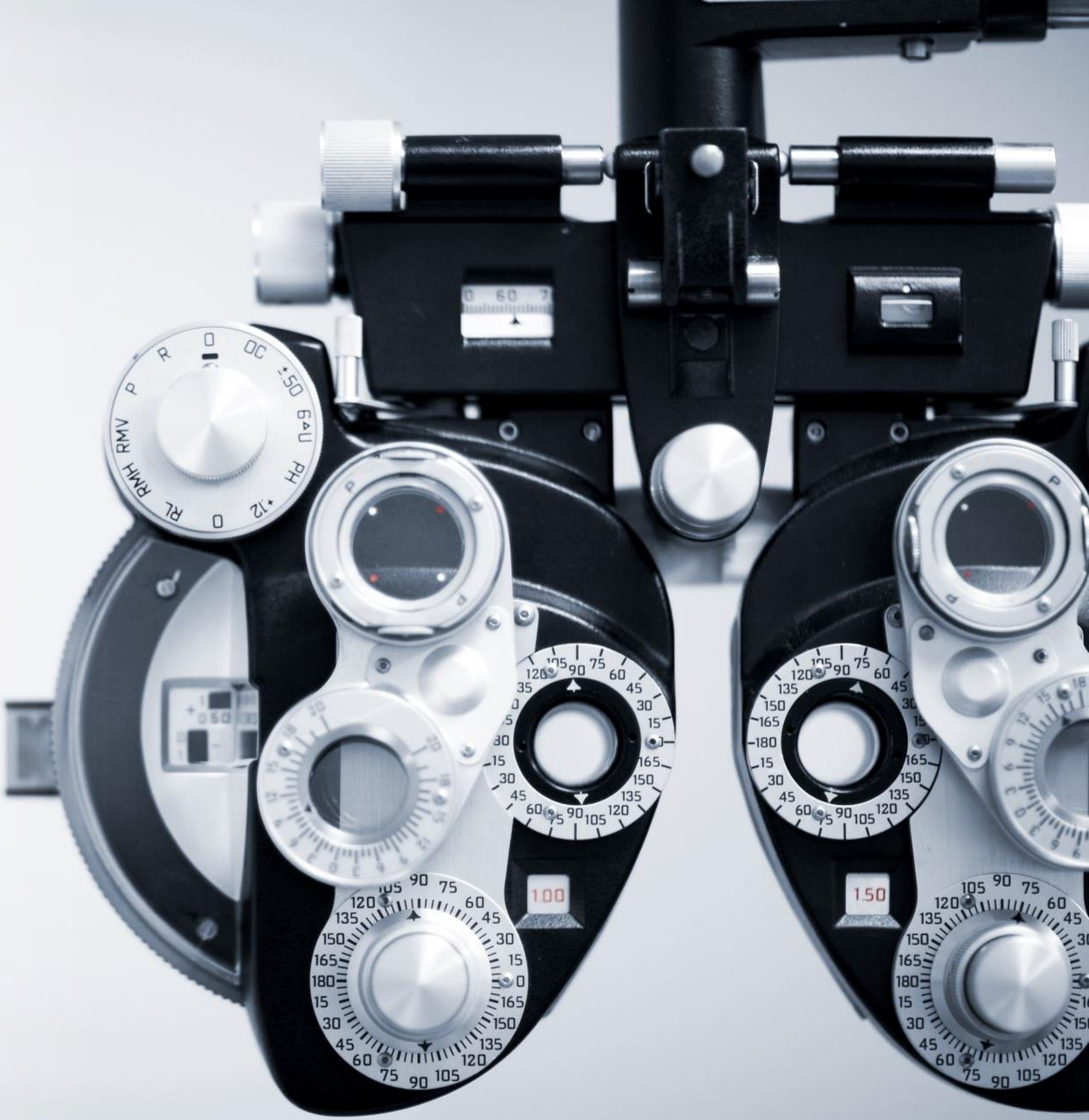
- Stragglers are ***transient***, primarily caused by degradation in computation (CPU contention & slow GPUs) and communication (network congestion)
- Computation stragglers are ***short-lived, less frequent***; communication stragglers are ***more frequent*** and last ***longer time***, causing ***more significant degradation***
- Large-scale training experienced ***both computation and communication stragglers***, causing significant throughput loss, potentially exceeding 90%

Agenda

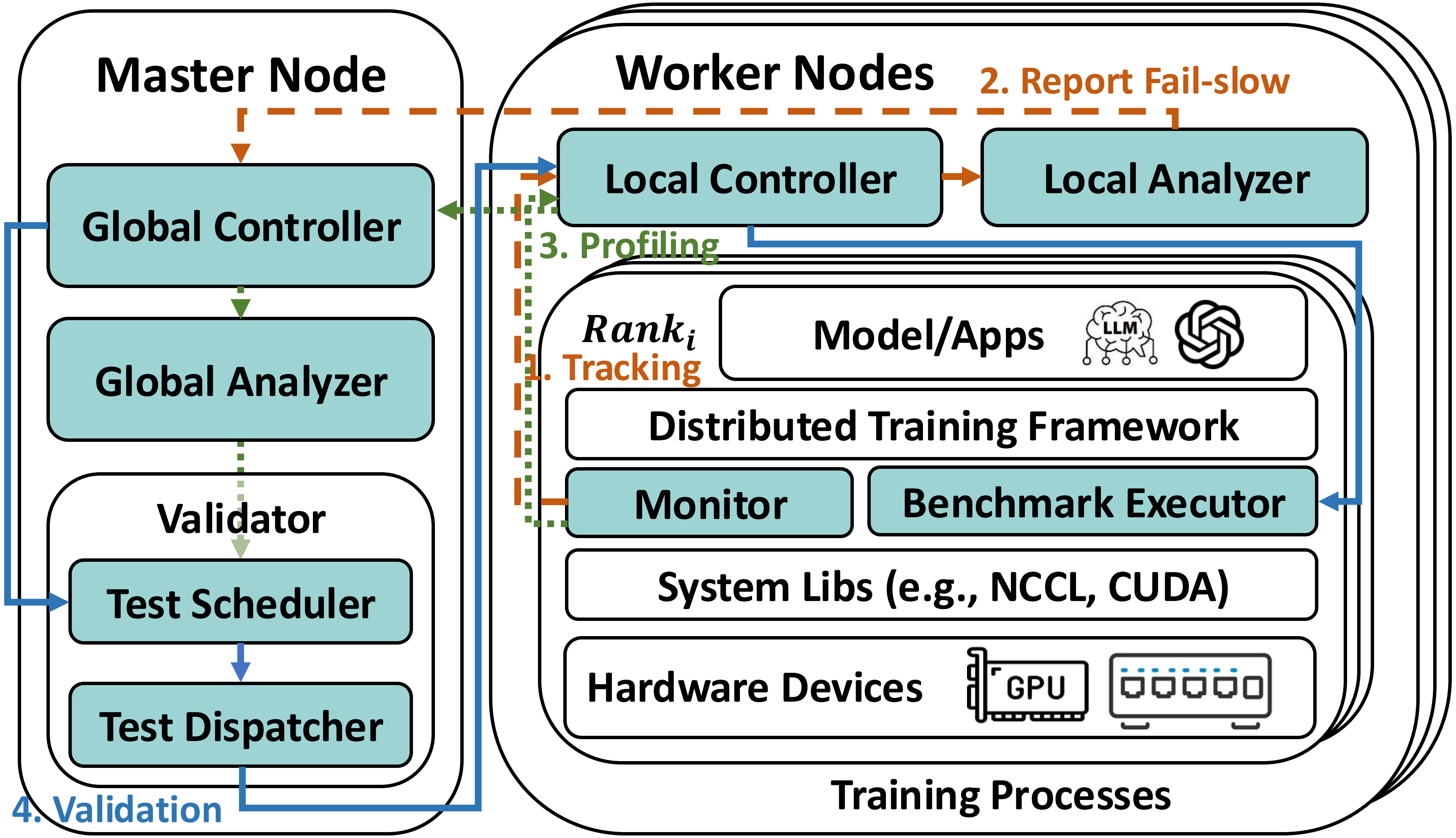
- Reliability issues in large-scale training.
- How do stragglers manifest in hybrid-parallel training at scale?
- How can stragglers be detected rapidly?
- How should stragglers be mitigated effectively?
- How do our detection and mitigation solutions perform?

Design Requirements

- Non-intrusive and framework-transparent
- Rapid and accurate
- Fully automated
- Lightweight, with minimum performance overhead



Overview of Greyhound-Detect System



Technical Challenges

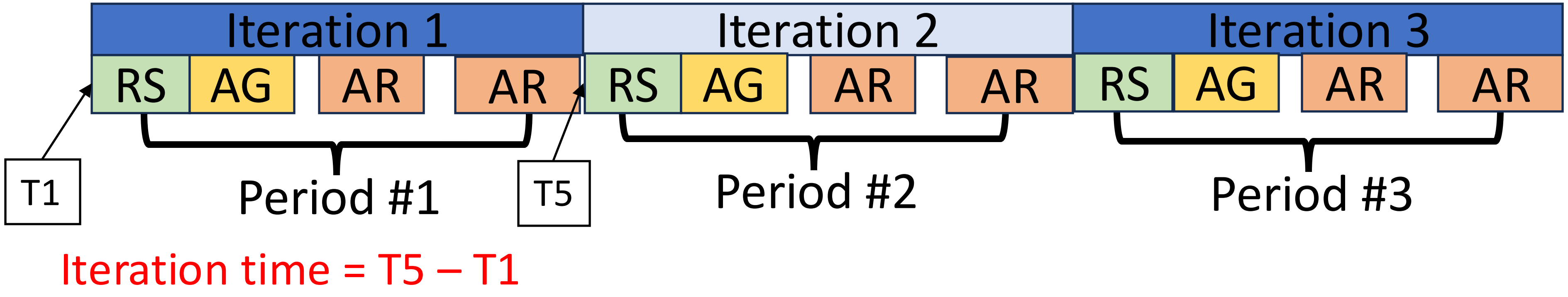
- **Challenge #1:** How to infer the iteration time without framework's cooperation?
- **Challenge #2:** How to detect the onset and termination of a straggler event?
- **Challenge #3:** How to profile the slow GPU or communication group? [In Paper]
- **Challenge #4:** How to locate the congested link within a group? [In Paper]

Non-Intrusive Iteration Time Inference

Challenge #1: How to infer the iteration time without framework's cooperation?

- Hook to NCCL calls and intercept Communication Ops via Linux's LD_PRELOAD
- Training is iterative, w/ periodic Communication Op patterns over iterations
- Identify periodic Op patterns via time-series analysis and infer the iteration period

Communication Ops: ReduceScatter (RS), AllGather (AG), 2*AllReduce (AR)



Detecting the Onset of a Slow Iteration

Challenge #2: How to detect the onset and termination of a straggler event?

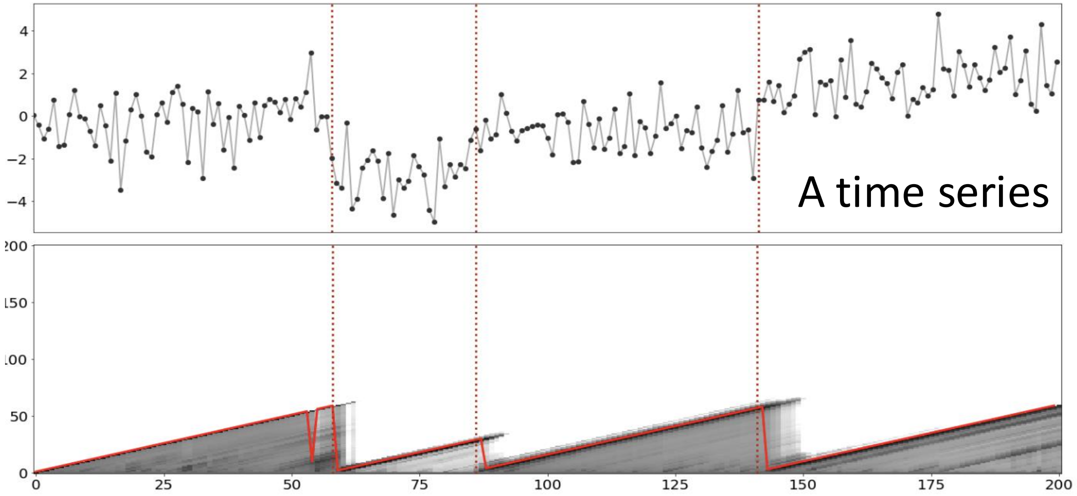
Bayesian online change-point detection (BOCD) + Verification to filter out false-positives

- A Bayesian method for online change-point detection
- Run length

$$r_t = \begin{cases} 0 & \text{if changepoint at time } t \\ r_{t-1} + 1 & \text{else.} \end{cases}$$

- Updating

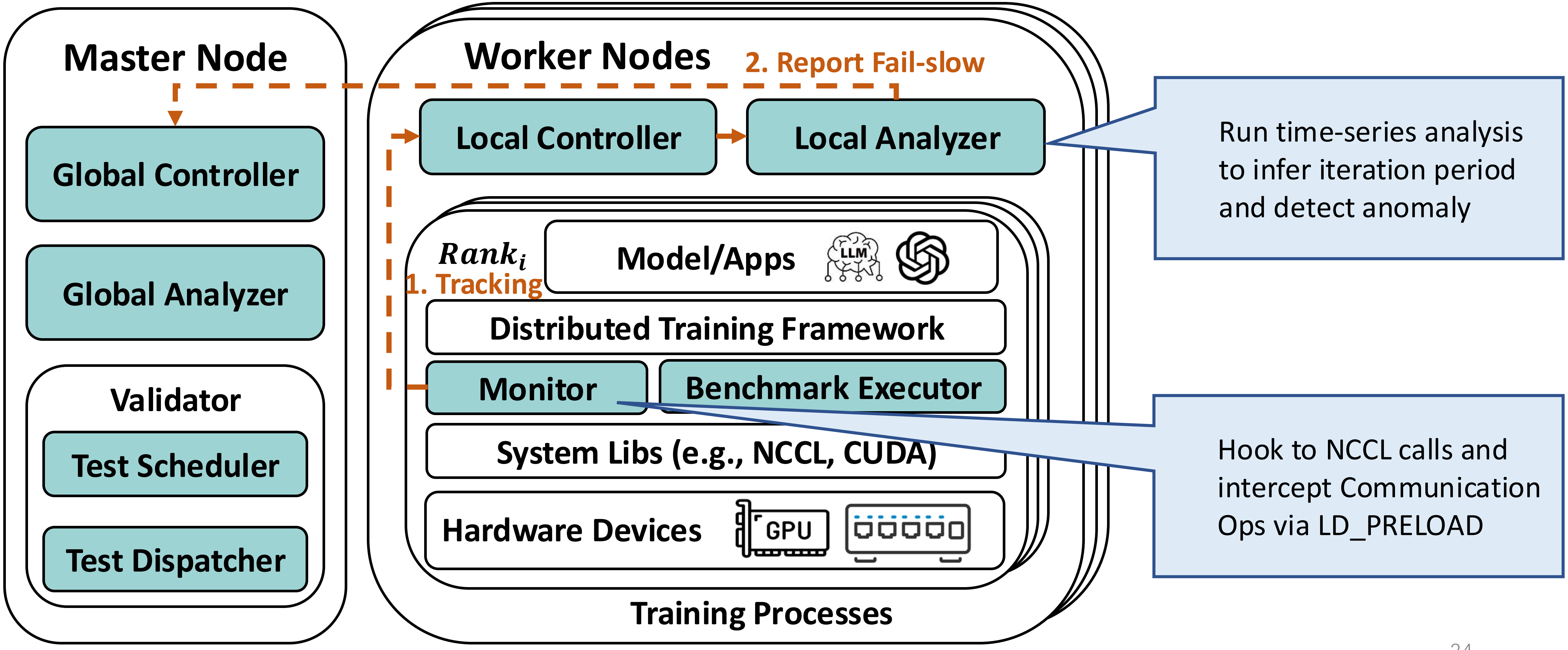
$$p(r_t, \mathbf{x}_{1:t}) = \sum_{r_{t-1}} \overbrace{p(x_t \mid r_t, \mathbf{x}^{(\ell)})}^{\text{UPM predictive}} \overbrace{p(r_t \mid r_{t-1})}^{\text{Changepoint prior}} \overbrace{p(r_{t-1}, \mathbf{x}_{1:t-1})}^{\text{Message}}.$$



The run length posterior at each time step; darker indicates higher probability

Credit: Adams and MacKay, https://gregorygundersen.com/blog/2019/08/13/bocd/

Non-Intrusive Straggler Inference



Agenda

- Reliability issues in large-scale training.
- How do stragglers manifest in hybrid-parallel training at scale?
- How can stragglers be detected rapidly?
- How should stragglers be mitigated effectively?
- How do our detection and mitigation solutions perform?

Design Requirements

- Reactive rather than predictive
  - Straggler occurrence and durations are unpredictable
- Online adjustment without restarting the training job
- Effective for both computation and communication stragglers

Design Space: the Four Mitigation Strategies

(S1) Do nothing: simply ignore fail-slow problems.

(S2) Adjust micro-batch distribution:

- Idea: assign less #micro-batches to slow DP groups → load balancing across DP groups

(S3) Adjust parallelism topology:

- Key insight: DP is more communication intensive than PP
- Idea: adjust parallelism, use congested links to serve PP traffic, and healthy links for DP traffic

(S4) Checkpoint and Restart: last resort, treat stragglers as failures

Strategy	Effectiveness		Action Overhead
	Slow Comp.	Slow Comm.	
S1: Ignore	No Effect	No Effect	No
S2: Adjust Microbatch	Mitigate	No Effect	Low
S3: Adjust Topology	Mitigate	Mitigate	Medium
S4: Ckpt-N-Restart	Eliminate	Eliminate	High

Design Space: the Four Mitigation Strategies

(S1) Do nothing: simply ignore fail-slow problems.

(S2) Adjust micro-batch distribution:

- Idea: assign less #micro-batches to slow DP groups → load balancing across DP groups

Optimal strategy depends on straggler impacts and duration, which cannot be known in prior

- Idea: adjust parallelism, use congested links to serve PP traffic, and healthy links for DP traffic

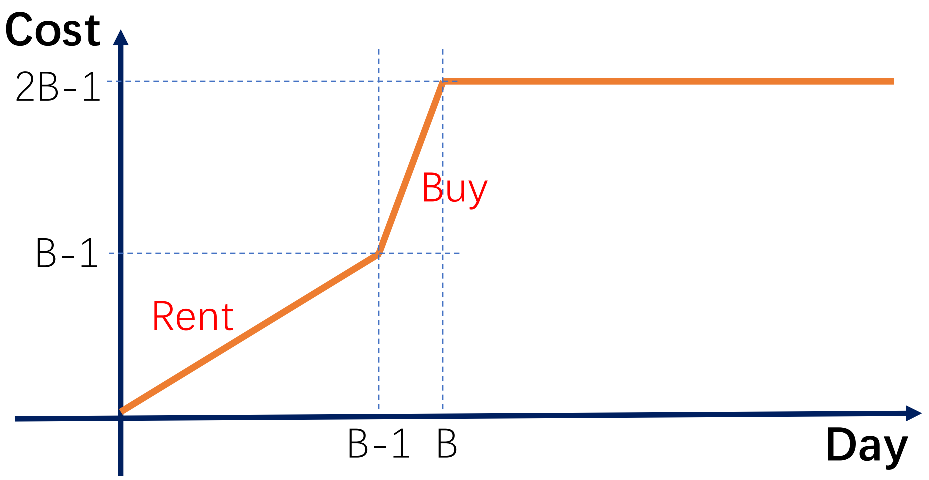
(S4) Checkpoint and Restart: last resort, treat stragglers as failures

Strategy	Effectiveness		Action Overhead
	Slow Comp.	Slow Comm.	
S1: Ignore	No Effect	No Effect	No
S2: Adjust Microbatch	Mitigate	No Effect	Low
S3: Adjust Topology	Mitigate	Mitigate	Medium
S4: Ckpt-N-Restart	Eliminate	Eliminate	High

The Ski Rental Problem

A skier goes to a ski resort with two choices: (1) **renting skis** for \$1 per day or (2) **buying skis** for \$B. The skier *has no idea how many days to ski* and needs to decide at the beginning of each day ***whether to rent or buy skis***.

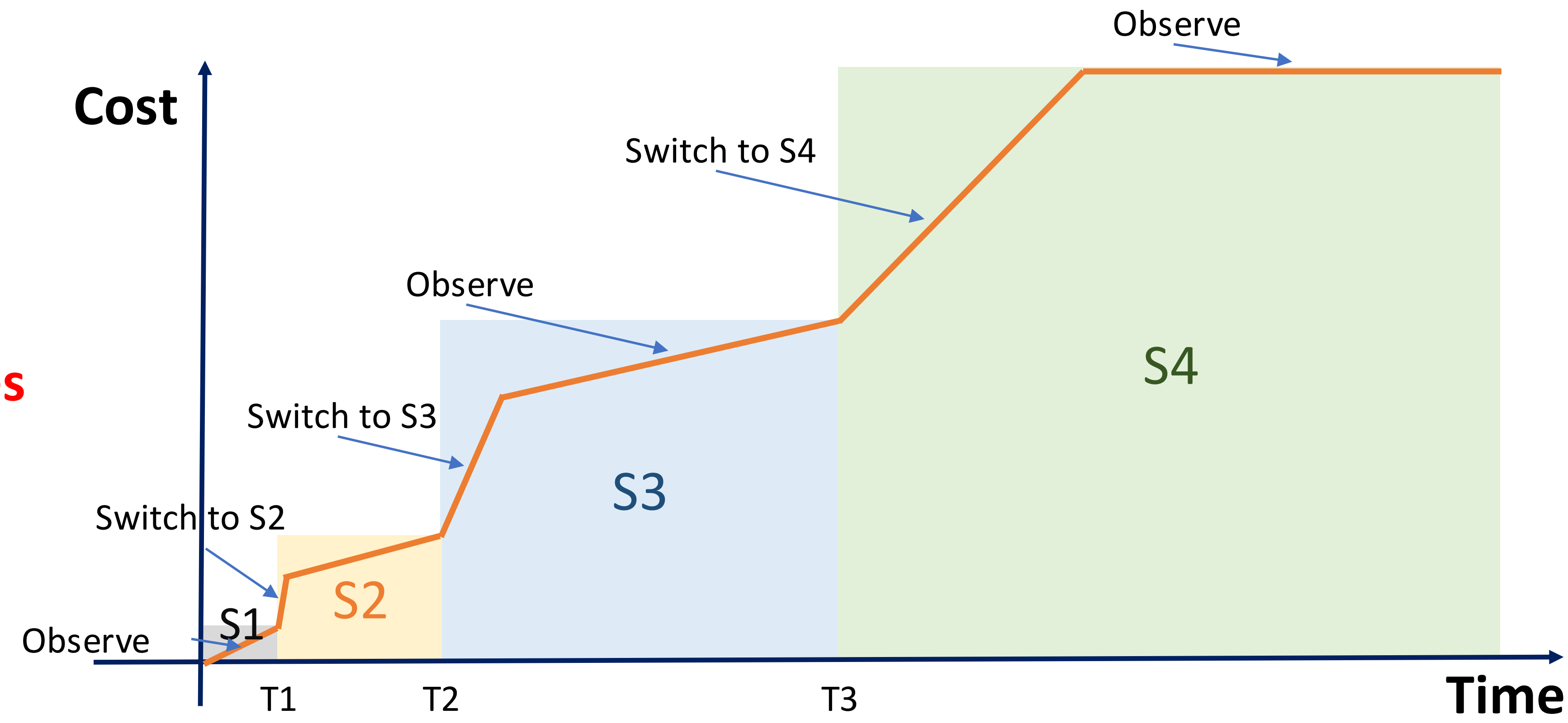
- **Optimal strategy:** Rent until realizing you should have bought, then buy
  - Rent on the first B-1 days, and then buy skis on the B-th day
  - The cost is **≤2x** of the ideal optimum, the best possible for a deterministic online algorithm



Multi-Level Straggler Mitigation

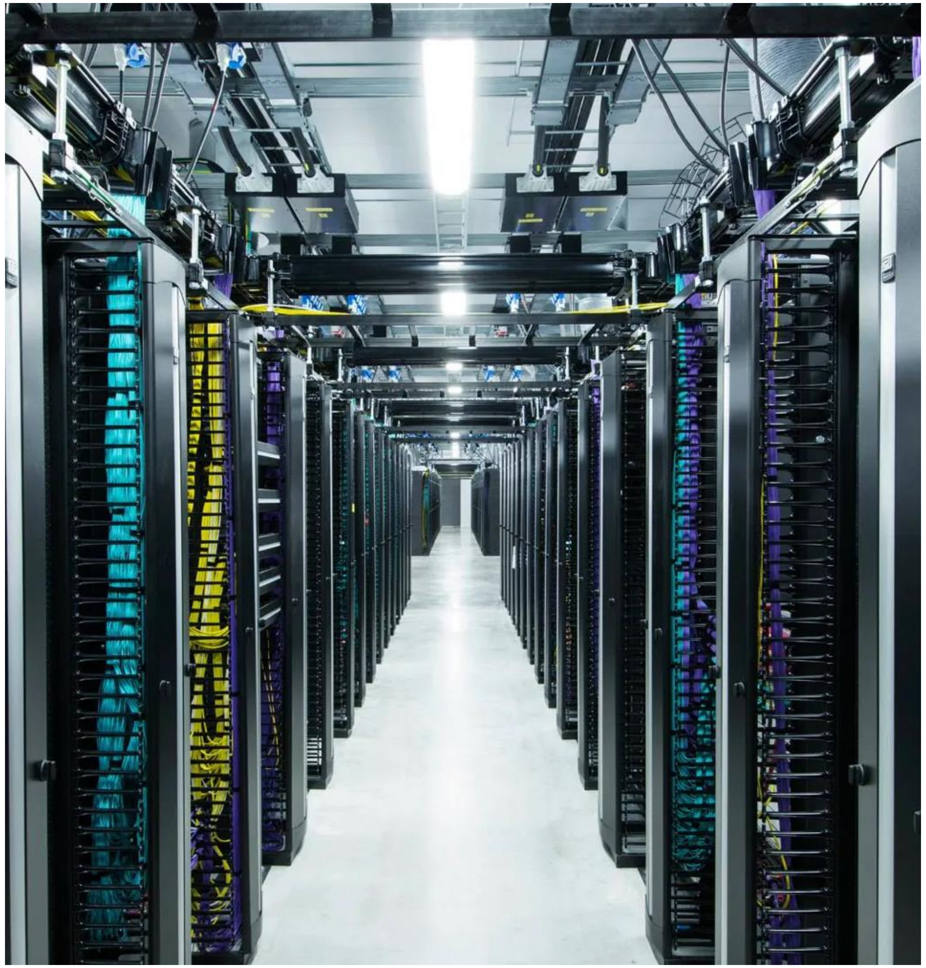
Starts with a low-cost strategy (S1) and **progressively switches** to more effective, yet more costly ones

Strategy	Effectiveness		Action Overhead
	Slow Comp.	Slow Comm.	
S1: Ignore	No Effect	No Effect	No
S2: Adjust Microbatch	Mitigate	No Effect	Low
S3: Adjust Topology	Mitigate	Mitigate	Medium
S4: Ckpt-N-Restart	Eliminate	Eliminate	High



Agenda

- Reliability issues in large-scale training.
- How do stragglers manifest in hybrid-parallel training at scale?
- How can stragglers be detected rapidly?
- How should stragglers be mitigated effectively?
- How do our detection and mitigation solutions perform?



Methodology

- **Testbed**
  - NVIDIA H800 SuperPOD, 400 Gbps IB
  - Up to 256 H800 GPUs in 8 DGX servers
  - Framework: Megatron-LM
- **Straggler injection**
  - *Slow computation:* throttle GPU frequency with nvidia-smi
  - *Slow communication:* launch communication-intensive jobs to create network congestion

How Accurate Is Detection?

Probing jobs: specially designed to detect slow computation and/or communication

- Type-A for slow comput.: 4x H800 on 1 node, GPT-2 11B, 2TP-2PP, 10K iterations
- Type-B for slow commun.: 8x A100 on 4 noes, GPT-2 7B, 2TP-4DP, 10K iterations

Manually verified the probing results via trace inspection

Type-A for slow computation (single node)

Algorithm	Accuracy↑ (%)	FPR↓ (%)	FNR↓ (%)
SlideWindow	99.5(390/392)	**0.0(0/386)**	25.0(2/8)
BOCD	77.8(305/392)	18.39(87/473)	**0.0(0/6)**
BOCD+V	**100.0(392/392)**	**0.0(0/386)**	**0.0(0/6)**

Type-B for slow communication (4-node)

Algorithm	Accuracy↑ (%)	FPR↓ (%)	FNR↓ (%)
SlideWindow	93.5(100/107)	1.5(1/65)	12.2(6/49)
BOCD	69.2(74/107)	34.0(33/97)	**0.00(0/43)**
BOCD+V	**99.1(106/107)**	**0.00(0/64)**	2.3(1/44)

Overhead: ~0.39% across all jobs, barely negligible

How Effective Is Mitigation?

Mitigating comput. stragglers (S2)

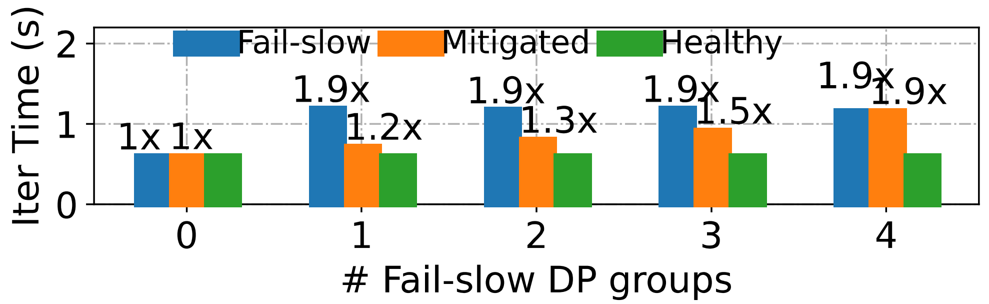
- Inject slow computations into 0-4 DP groups in a 4-DP training job
- Overhead: <30s even #DP=512.

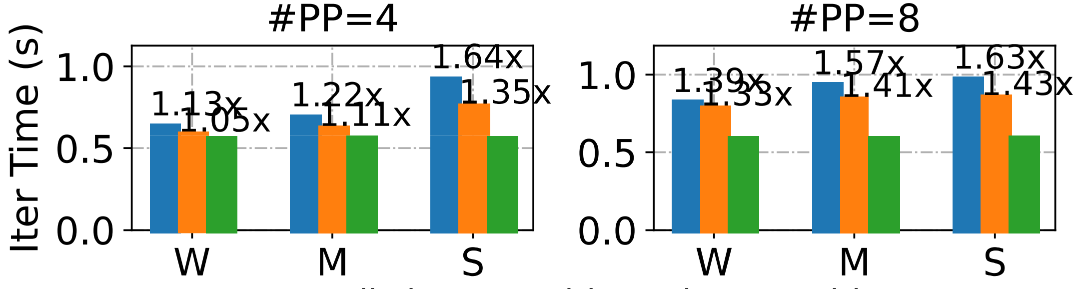
Mitigating commun. stragglers (S3)

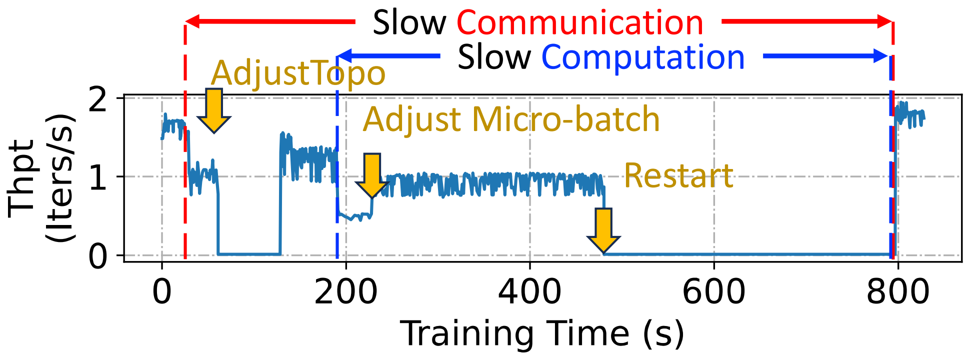
- Inject weak, medium and strong communication stragglers into 16-GPU training jobs w/ 4 and 8 PP stages
- Overhead: <50s in our cluster, 6.72x faster than ckpt-n-restart.

Slow commun. + slow comput.

- 16-GPU training w/ (4DP, 4PP)





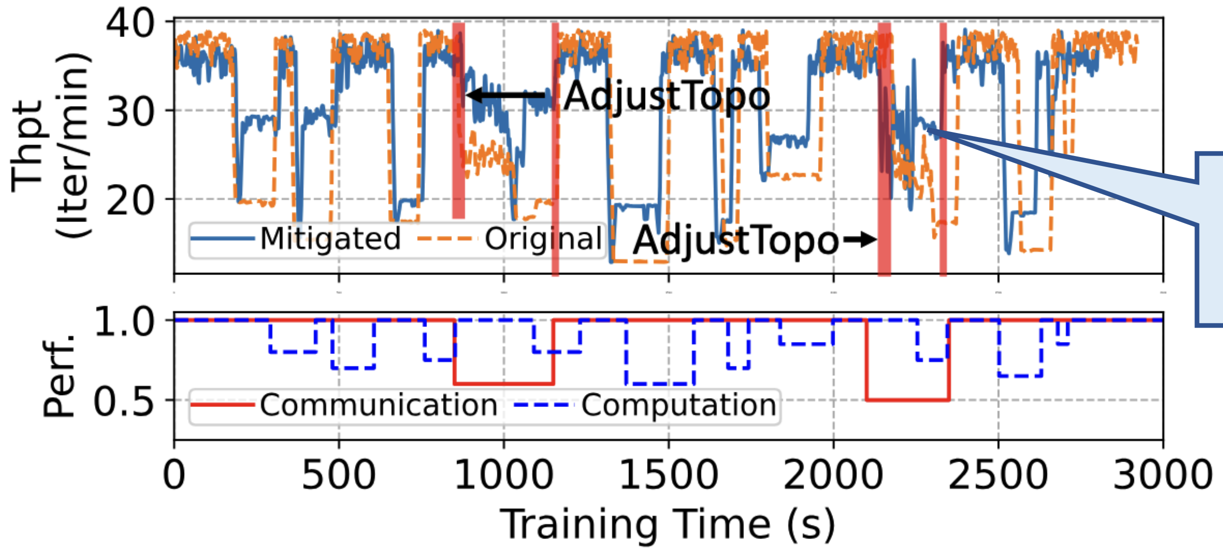


How Does Greyhound Perform at Scale?

- Training GPT2-40B on 256 GPUs using (8TP, 16DP, 2PP)
  - Inject 2 communication stragglers and 8 computation stragglers of varying severity

Comm. Slow	Comp. Slow	Detect Acc.	Avg. Reaction
Medium: 2	*Weak*: 3 *Medium*: 5 *Severe*: 2	100%	*BOCD*: 3.7s *Prof.&Val.*: 6.86s *Total*: 10.56s

Identify stragglers in 10s



End-to-end performance: 1.58x faster

Conclusion

- First comprehensive characterization study of straggler problems for LM training
  - Stragglers are ***transient***, ***frequent***, and can result in ***significant training slowdown***
  - Computation stragglers are ***short-lived, less frequent***; communication stragglers are ***more frequent*** and last ***longer time***, causing more significant degradation
- Straggler detection
  - ***Non-intrusive, rapid, accurate, and lightweight***
- Effective multi-level straggler mitigation
  - Four possible mitigation strategies
  - ***Start w/ a low-cost one and progressively switches to more effective, yet more costly ones***

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Alibaba Group 阿里巴巴集团

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED