Efficient Fault-Tolerant Training

MLSys Seminar

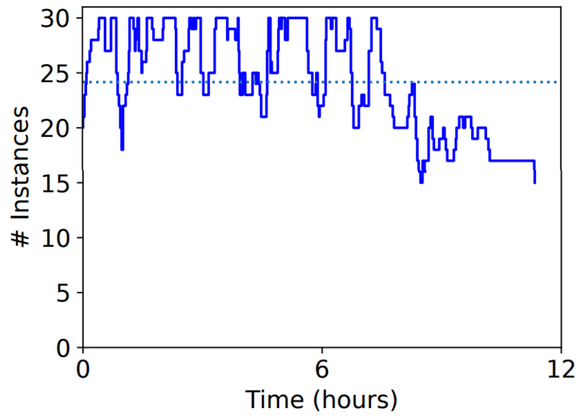
Nov 26, 2024

About Me

- Xiwen Min
- Master of Computer Science, Courant
- BEng @Shanghai Jiao Tong University

Failures in Large Scale Training

- Machine failures are common in large scale training
  - LLaMA team: 466 job interruptions during a 54-day snapshot period of pre-training
  - Large scale training clusters at Microsoft see a failure every ~ 45 minutes
  - OPT team: 178,000 GPU-hours were wasted due to malfunctions, 100+ host restart
  - With hybrid parallelism, one failure can affect the whole grid
- Training on Preemptive Instances suffer from frequent unpredictable preemptions



<= traces of AWS EC2 P3 instance availability

Recover from Failures

- Activate spare machines loading model from/preloaded with checkpoint
  - Restarting from the latest checkpoint can be slow
  - Expensive
- Drop the entire data parallel group and continue training
  - Underutilize system resources, 1/DP drop in throughput
  - Global reconfiguration is slow

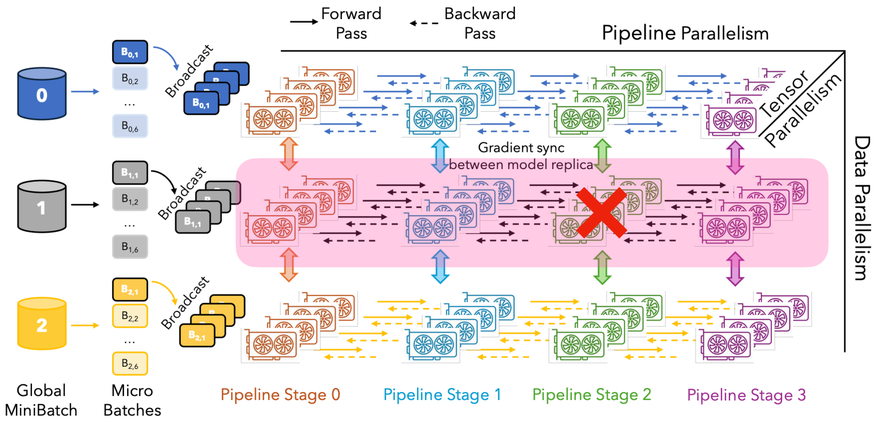
In the presence of (potentially frequent) failures, the speed of recovery is important.

Without spares, how to recover **fast** from failures?

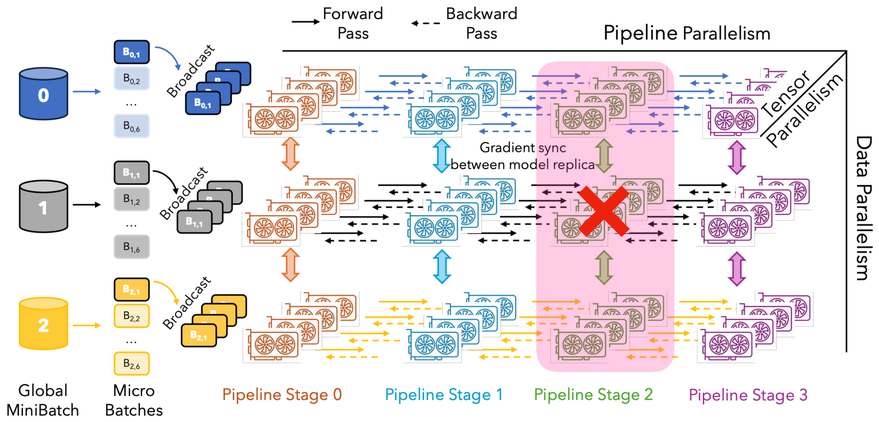
Intrinsic Redundancy in DP

- Instead of restarting from checkpoints, the model can be copied from available DP peers.
- Available nodes are partly (not globally) reconfigured to take over the workload of the failed node.

We will cover two ways to redistribute the workload of a failed machine:



Oobleck (SOSP'23)



ReCycle (SOSP'24)

Oobleck

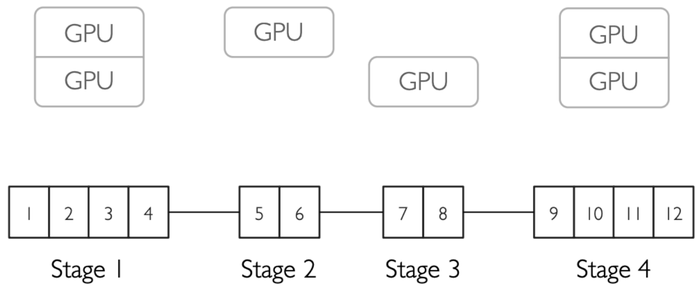
Overview

- Designed for general fault-tolerant scenarios
  - Both common machine failure pattern and preemptive spot instance pattern
- Guarantees tolerance of f failures without restarting from checkpoints
- Uses heterogeneous pipelines instantiated from ***pipeline templates***
  - DP=3 -> 3 pipelines, each may contain different numbers of nodes and stages

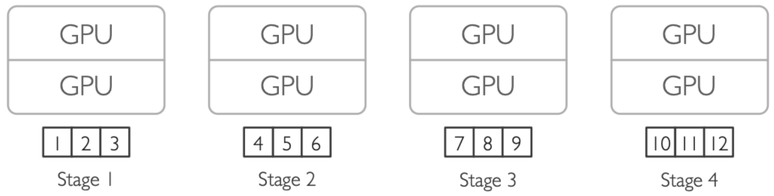
Oobleck

An Example of Failure Recovery

- A **Pipeline Template** is a pre-generated single pipeline execution specification for specific number of nodes



A Pipeline Template for 3 Nodes



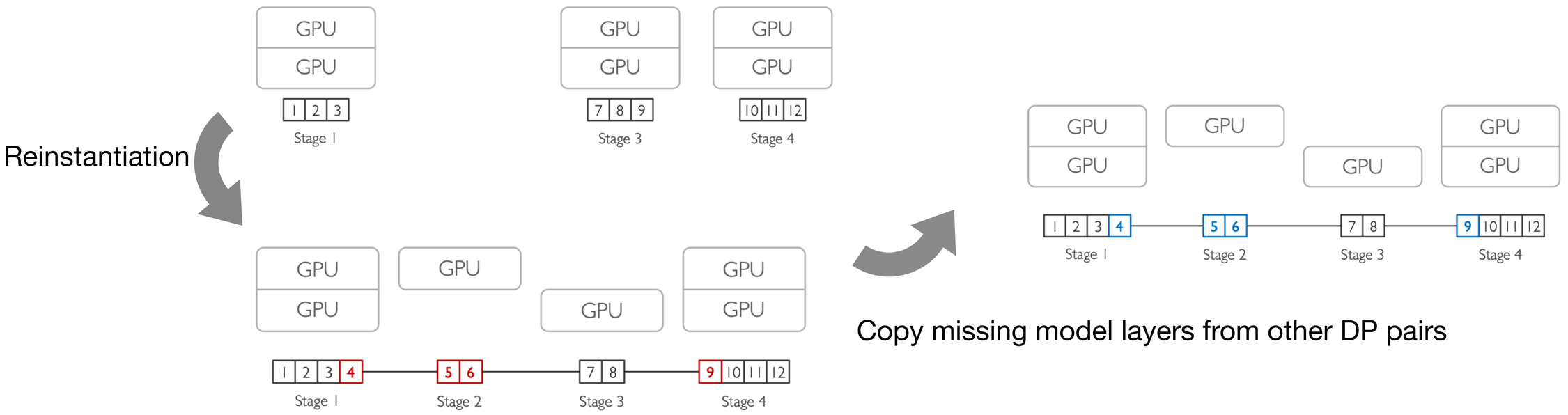
A pipeline instantiated from the template for 4 nodes

- Given a set of pipeline templates, the system and be instantiated with a combination of them
  - e.g. If the set of templates is {3, 4} as shown above, a system of 10 nodes, DP=3 can be instantiated with two 3-node pipelines and one 4-node pipeline

Oobleck

An Example of Failure Recovery

- When one node fails in an N-node pipeline, a new pipeline of (N-1) nodes is instantiated



Copy missing model layers from other DP pairs

Oobleck

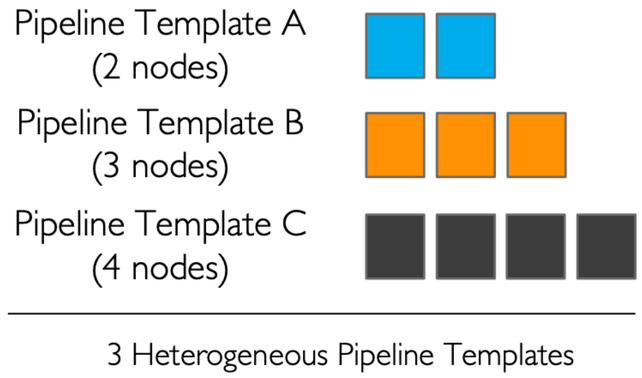
Details

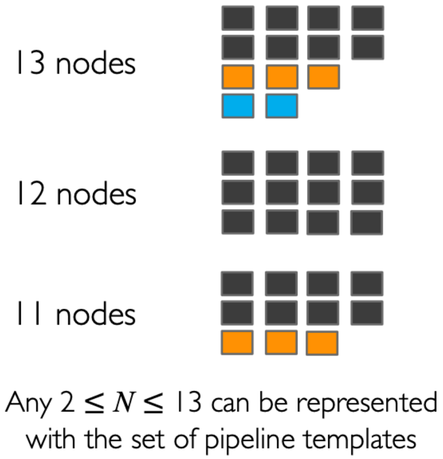
- How to determine such a set of templates?
  - How many templates? How many nodes for each template?
- Given a set of templates, how to determine the combination that results in max throughput?
- How to guarantee load balance of different pipelines?

Oobleck

Determine Templates: Node Specification

- For N available nodes, enumerate all templates with 2, 3, …, N nodes? Not necessarily.
- Frobenius Problem: Given $\{x_1, x_2, \ldots, x_m\}$, what's the largest number g that cannot be represented as a linear combination of the m numbers?
  - Find $\{x_1, x_2, \ldots, x_m\}$ s.t. $g < N - f$, we can utilize all nodes in the presence of f failures

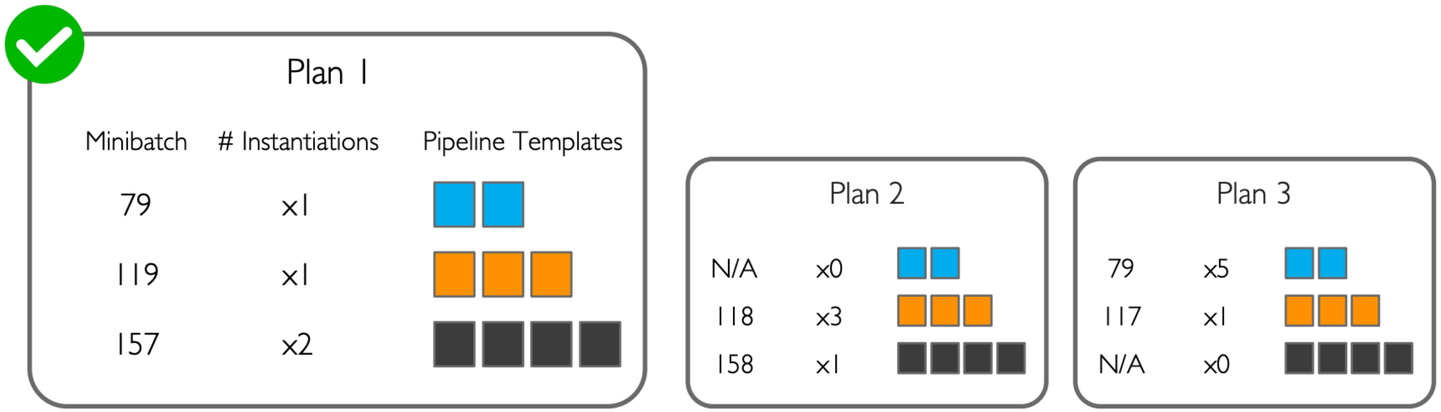




Oobleck

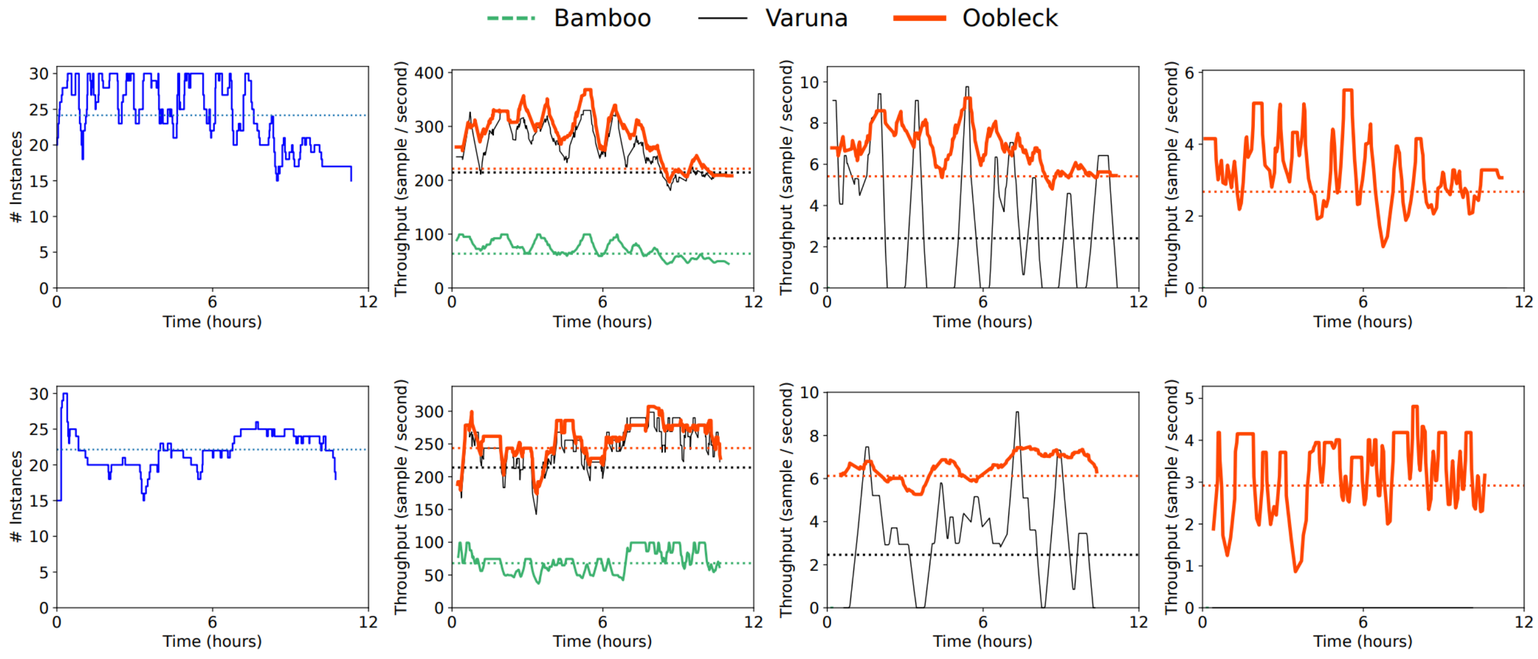
Determine The Combination & Load Balance: Instantiation

- For each combination, to minimize straggler effects, we need to assign different number of micro batches to different pipelines
- Given all enumerated possible plans of combination and corresponding per pipeline workload, we can estimate iteration time
  - Pick the best plan



Oobleck

Evaluation



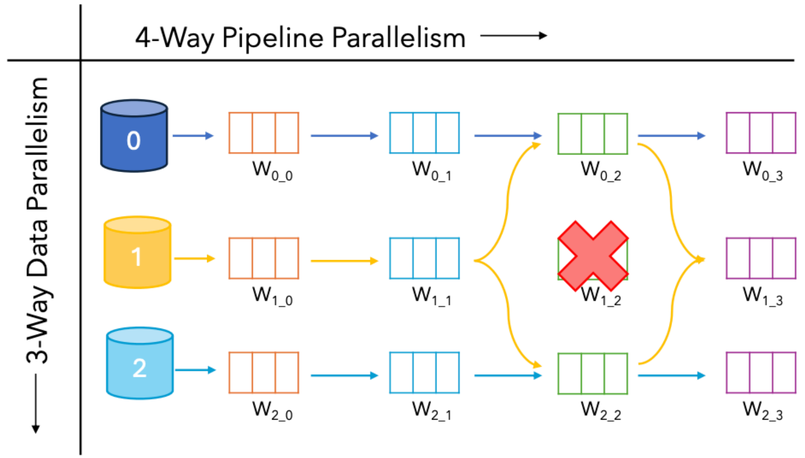
(a) *Node availability* **(b)** *BERT-Large* **(c)** *GPT-3 2.7b* **(d)** *GPT-3 6.7b*

Table 2. *Throughput (samples/s) with different frequency of failures. Bamboo was not able to run any GPT-3 model due to lack of memory (OOM).*

Models	BERT-Large			GPT-2			GPT-3 Medium			GPT-3 2.7b			GPT-3 6.7b		
Failure Frequency	6h	1h	10m	6h	1h	10m	6h	1h	10m	6h	1h	10m	6h	1h	10m
Bamboo	77.04	75.60	69.84	17.47	17.13	16.01									
Varuna	259.57	245.39	168.15	86.42	83.94	69.67	29.52	27.61	20.71	7.27	6.41	0.36	4.02	2.91	0.12
Oobleck	287.10	286.28	282.11	85.59	85.42	84.80	29.30	29.21	28.70	7.29	7.23	6.89	4.33	4.22	3.55

ReCycle

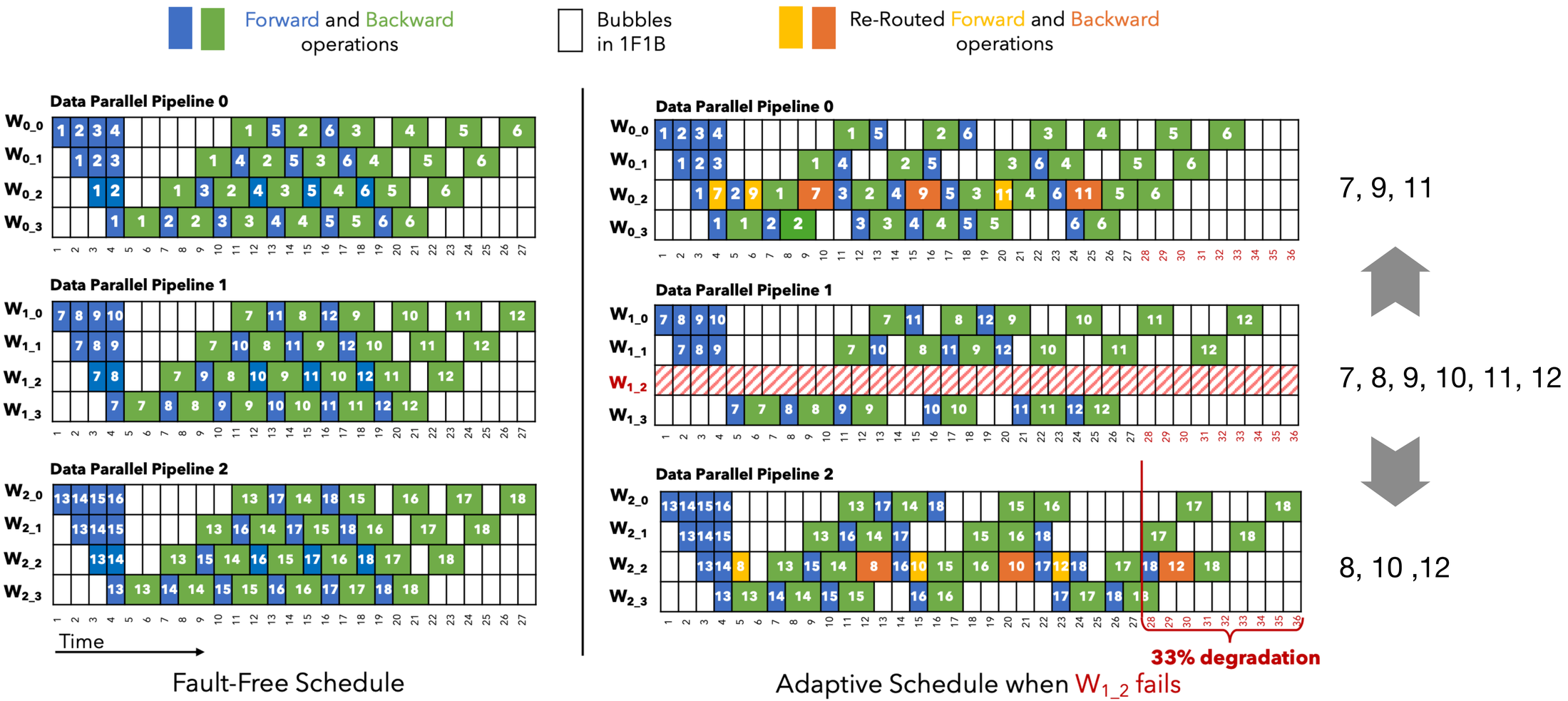
Overview



- Designed for general fault-tolerant scenarios
- Guarantees continuous training in the presence of (DP - 1) failures
- Unlike Oobleck that reshards the model when reinstantiate new templates, ReCycle does not move model. Instead, it forwards the workload of a failed node to its DP peers' **pipeline bubbles** of 1F1B schedule
  - $3 \times (PP - 1) \times DP$ bubbles per pipeline stage during warm-up & cool-down
  - 405 billion parameter LLaMA-3
    - TP=8, PP=16, DP=64 => $(3 \times (16 - 1) \times 64) = 2880$ bubbles per iteration
    - BSZ=2048 => $(2048/64) = 32$ samples per DP

ReCycle

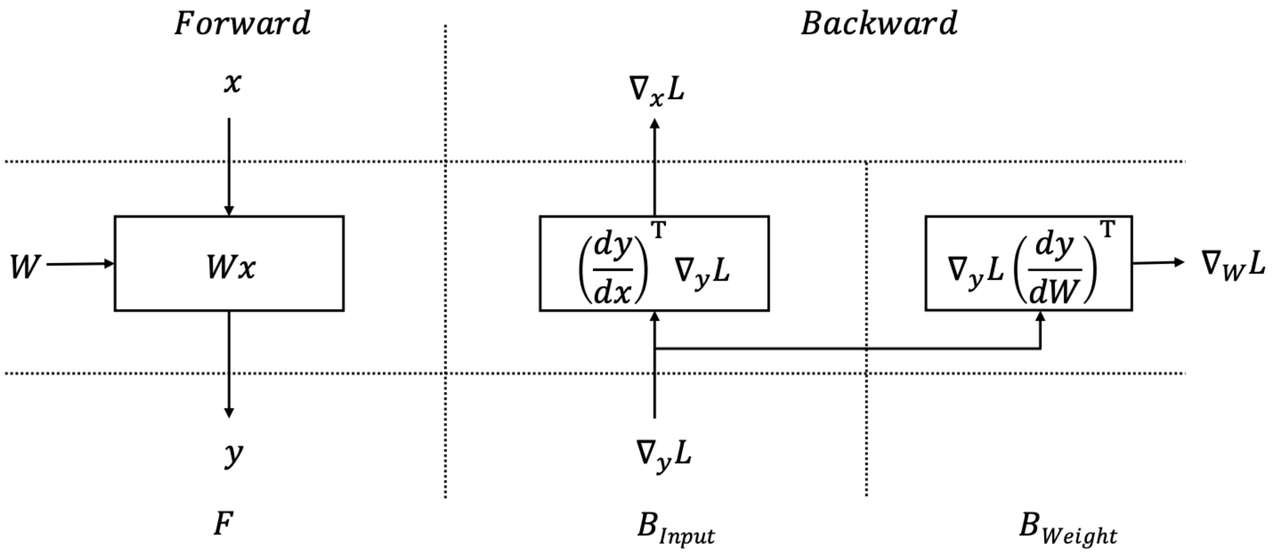
Attempt #1: Adaptive Pipelining



ReCycle

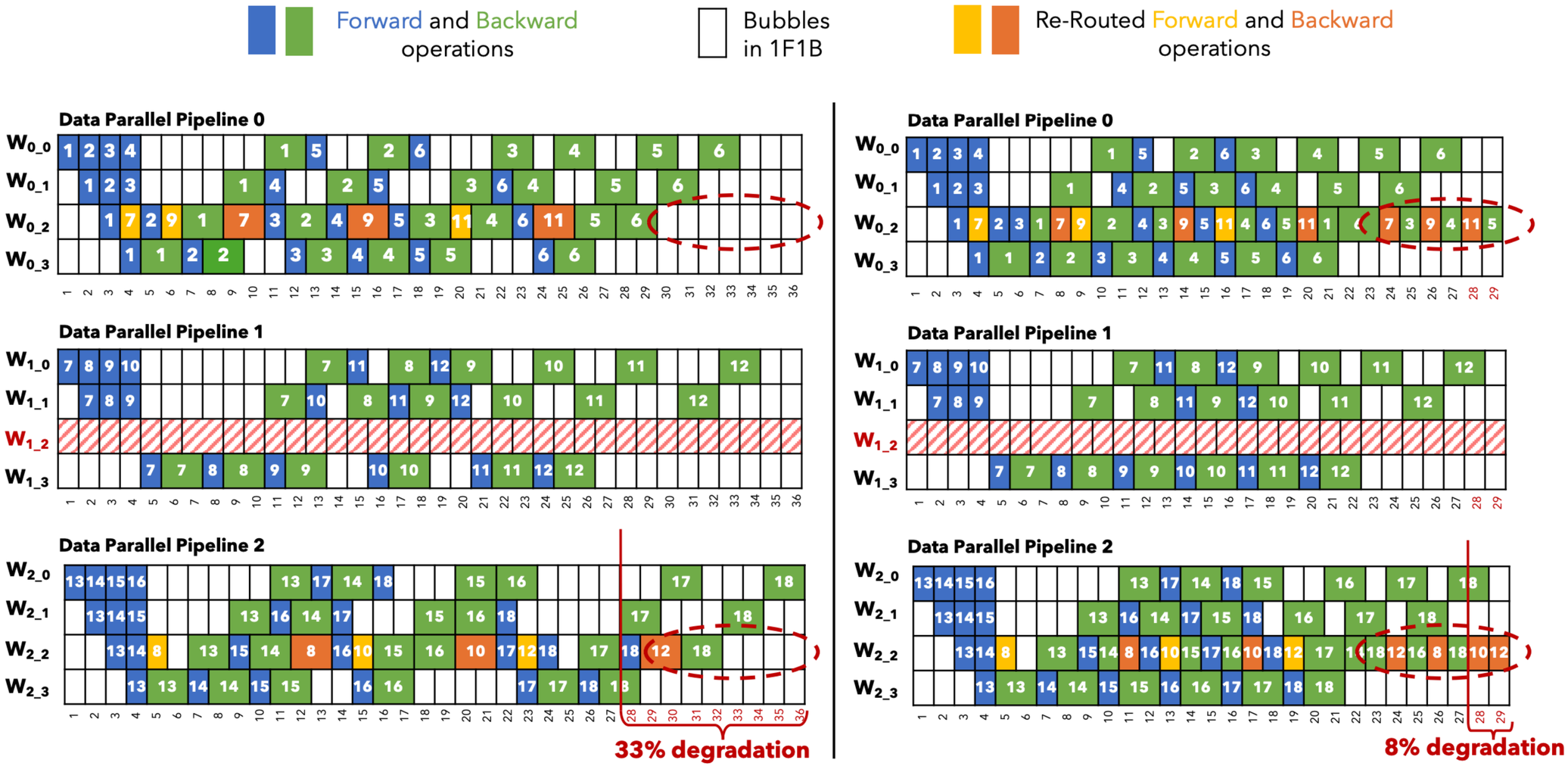
Attempt #2: Decoupled Backdrop

- Gradient wrt weights has not downstream dependency constraints
  - Split the backward path into two separated operations, defer the computation of gradients wrt weights to the end of iteration



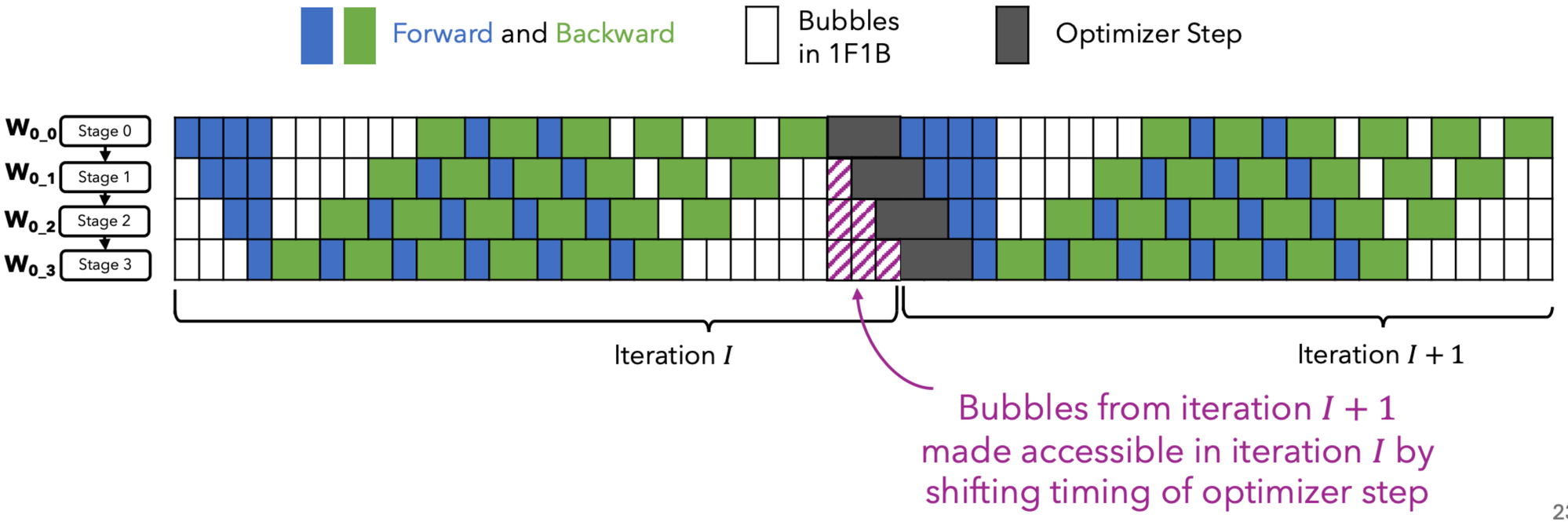
ReCycle

Attempt #2: Decoupled Backdrop



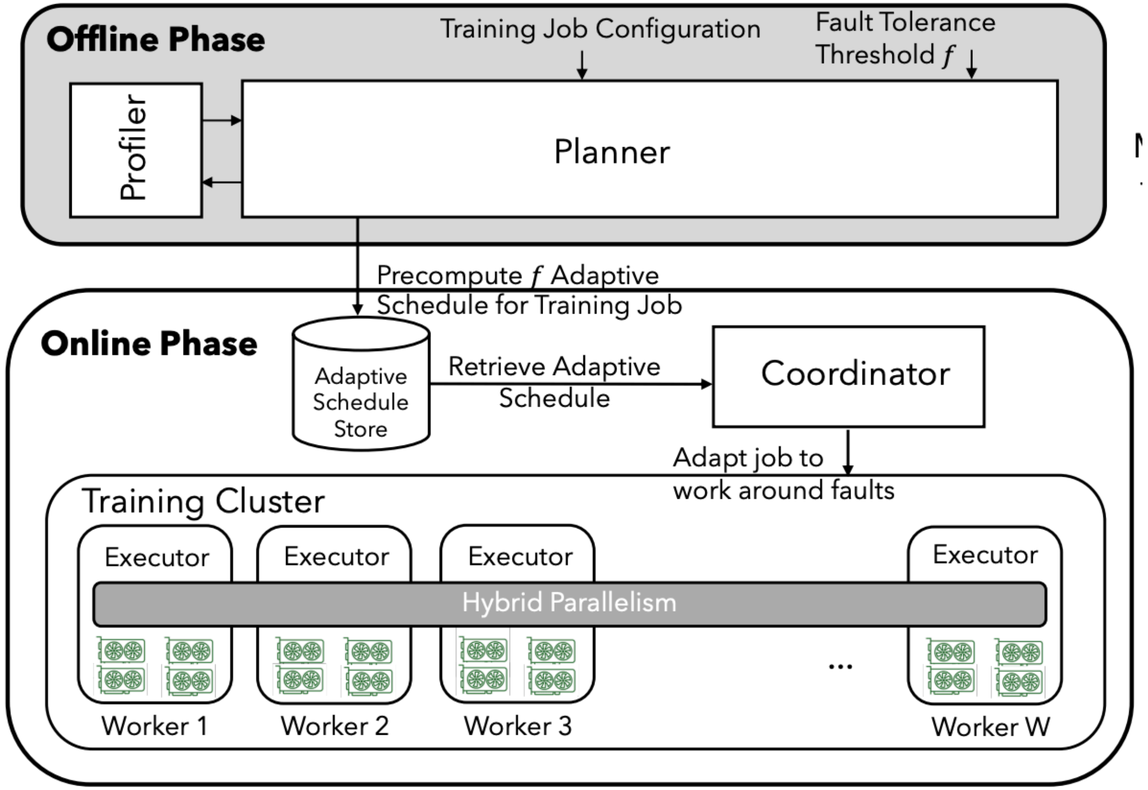
ReCycle

Attempt #3: Staggered Optimizer



ReCycle

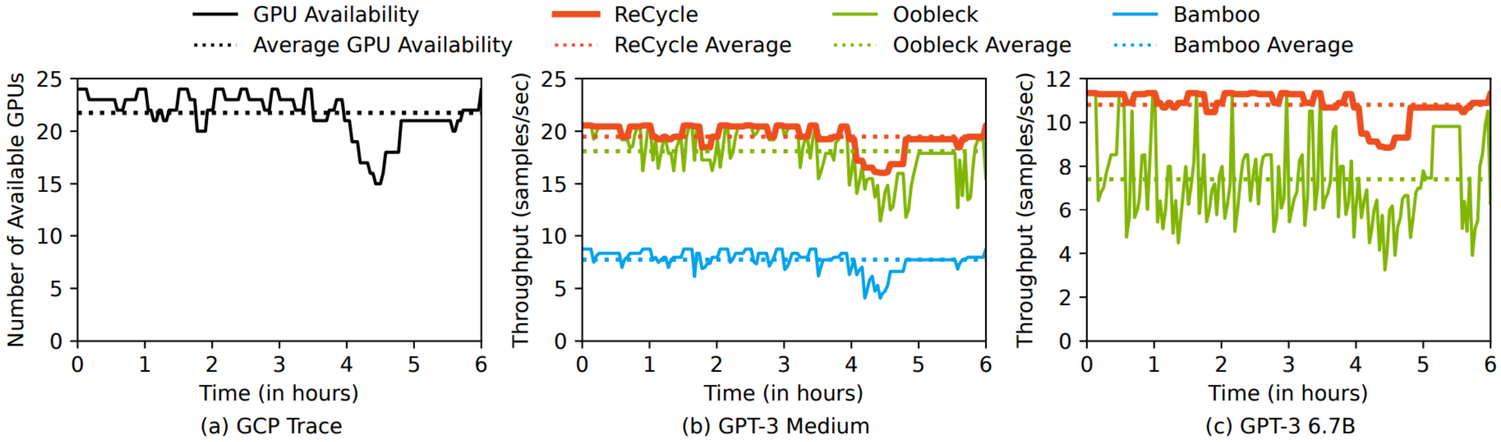
System Prototype



- Given a fault tolerance threshold, for each possible failure distribution, the Planner computes the schedule of micro batch re-routing with least additional time slots
  - Dynamic Programming
  - Mixed Integer Linear Programming

ReCycle
Evaluation

Systems	GPT-3 Medium			GPT-3 3.35B			GPT-3 6.7B		
Failure Frequency	6h	2h	30m	6h	2h	30m	6h	2h	30m
Fault-Free DeepSpeed [60]		27.58			14.87			5.33	
Bamboo [67]	19.47	18.98	15.24	OOM	OOM	OOM	OOM	OOM	OOM
Oobleck [29]	27.26	25.37	19.47	14.55	13.44	9.78	4.98	4.65	2.78
ReCycle	27.27	25.42	22.27	14.59	14.17	12.63	5.17	4.85	3.53



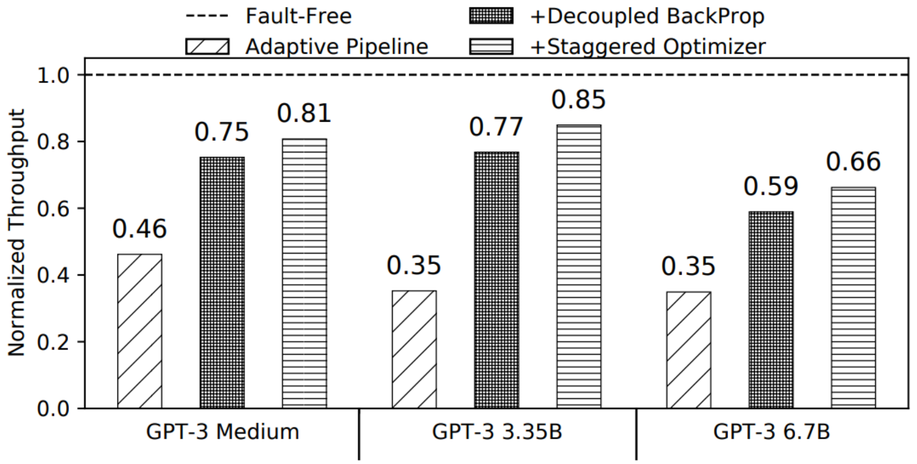


Figure 11. The contribution of the three optimization techniques to ReCycle's training throughput.

Related Work: Bamboo (NSDI'23)

- Designed for preemptive instances
- Not spares, but introduce **Redundant Computation**
  - Each node performs both *normal computation* over its own layers, and *redundant computation* over its successor's layers