天津大学

ReCycle: Resilient Training of Large DNNs using Pipeline Adaptation

Swapnil Gandhi
gandhis@stanford.edu
Stanford University

Mark Zhao
myzhao@cs.stanford.edu
Stanford University

Athinagoras Skiadopoulos
askiad@stanford.edu
Stanford University

Christos Kozyrakis
kozyraki@stanford.edu
Stanford University

汇报人：黄文豪、陈奕池、王衍杰

天津大学智能与计算学部

目录

1 Introduction

2 Motivation

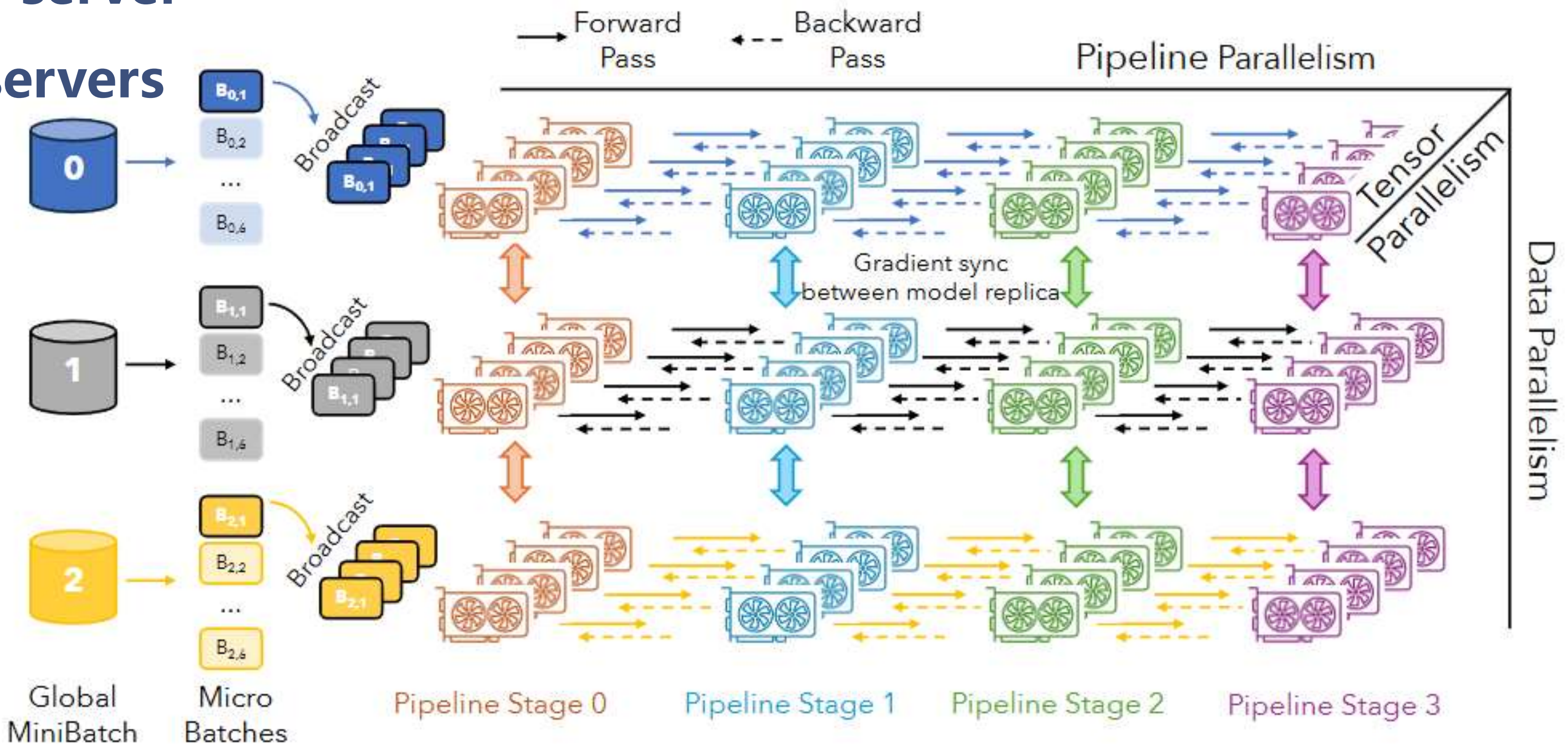
3 Design

4 Evaluation

5 Conclusion

Training background

- **Large scale training**
  - Llama-3 was trained on 15 trillion tokens, using two clusters of 24K GPUs
- **Hybrid-parallelism training**
  - TP within a multi-GPU server
  - PP across multi-GPU servers
  - DP across pipelines



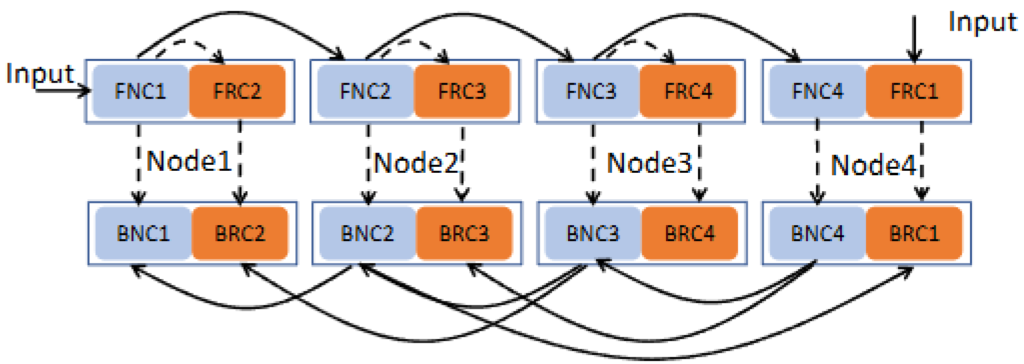
Fault during training

- **High cost of faults**
  - Microsoft's training cluster fails about every 45 minutes
  - Meta encountered over 100 hardware failures during OPT-175B training, losing 178,000 GPU hours
- **Fault handling**
  - Error Detection
  - Checkpoint
  - Fault tolerant training

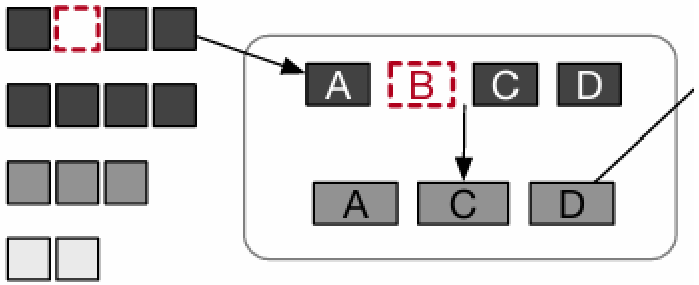
Related Works

Fault-tolerant training

- Bamboo (NSDI 23): redundant computation----- low throughput

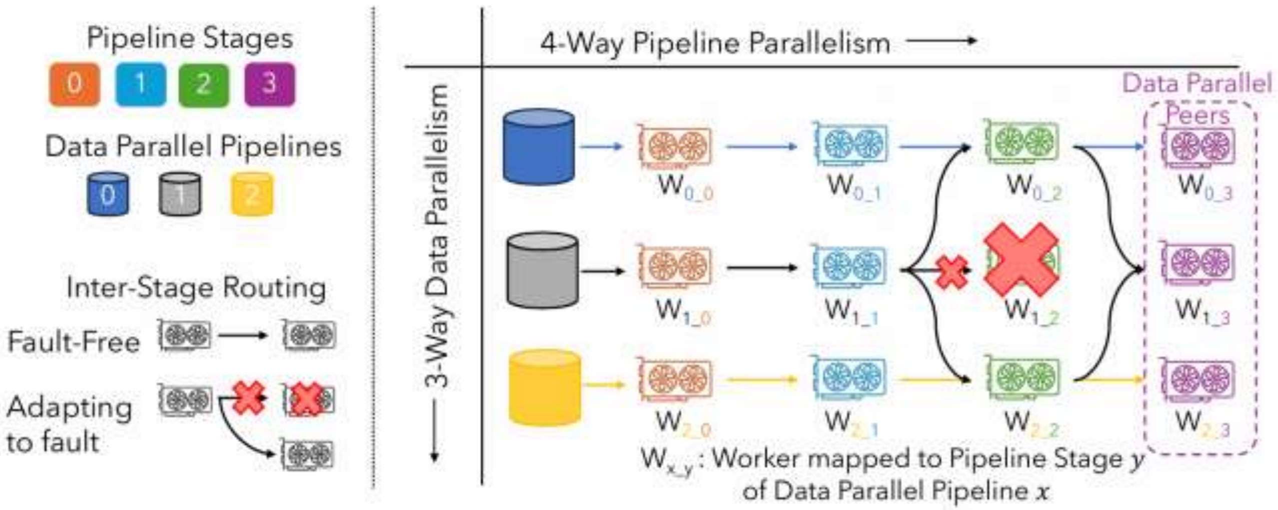


- Oobleck (SOSP 23): re-configure parallel scheme----- suspend overhead



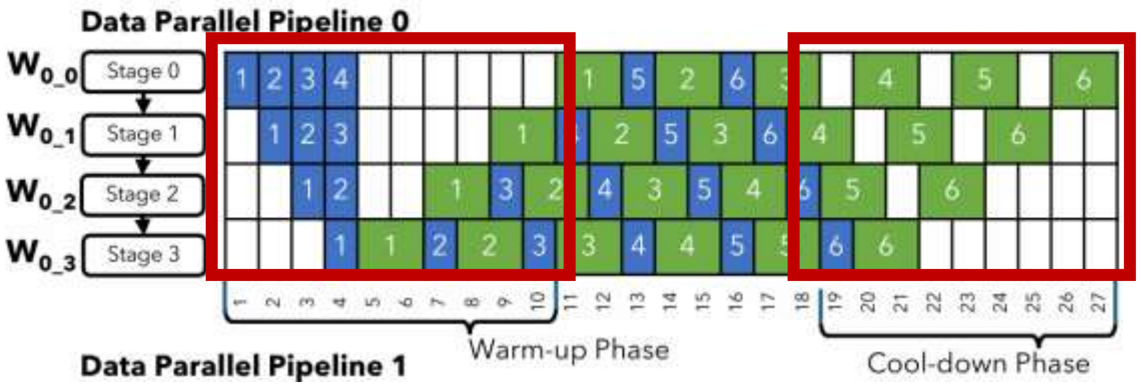
Motivation

1. Peer nodes have the same parameters in data parallelism



Reroute: No need to re-shuffle model parameters.

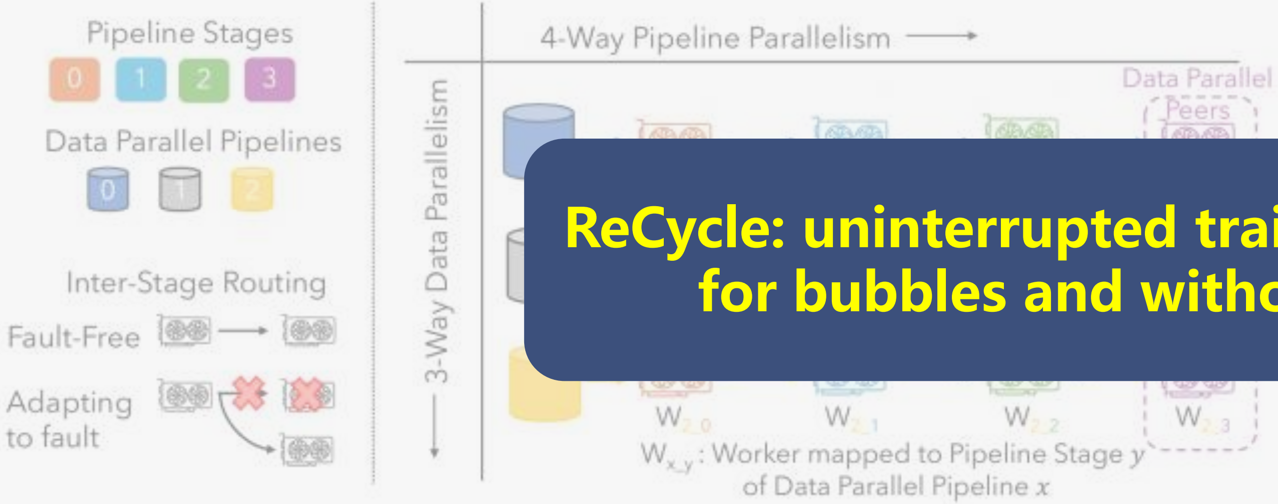
2. Bubbles in the pipeline parallelism



The warm-up and cool-down phases contain bubbles due to the sequential dispatch of micro-batches.

Motivation

1. Peer nodes have the same parameters in data parallelism



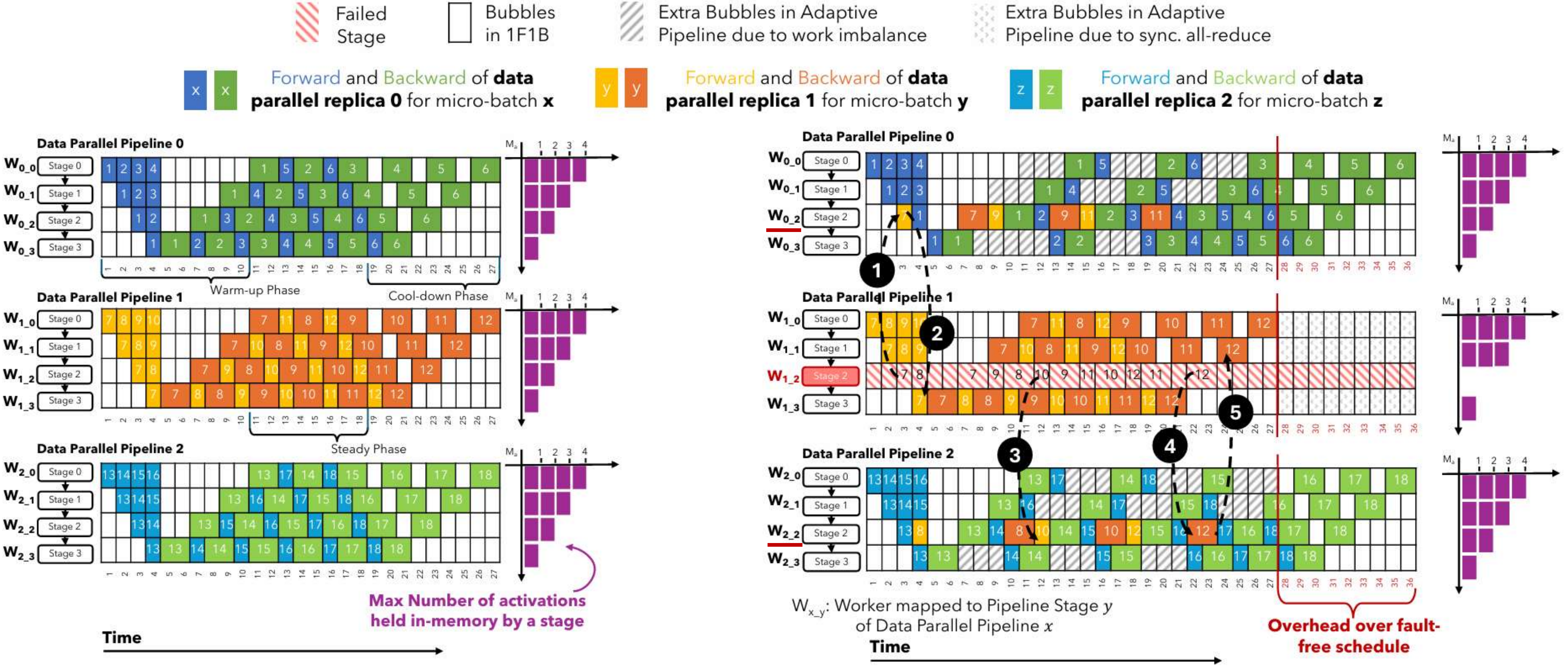
Reroute: No need to re-shuffle model parameters in case of a fault.

2. Bubbles in the pipeline parallelism

The warm-up and cool-down phases contain bubbles due to the sequential dispatch of micro-batches.

ReCycle: uninterrupted training with re-routing for bubbles and without re-shuffling.

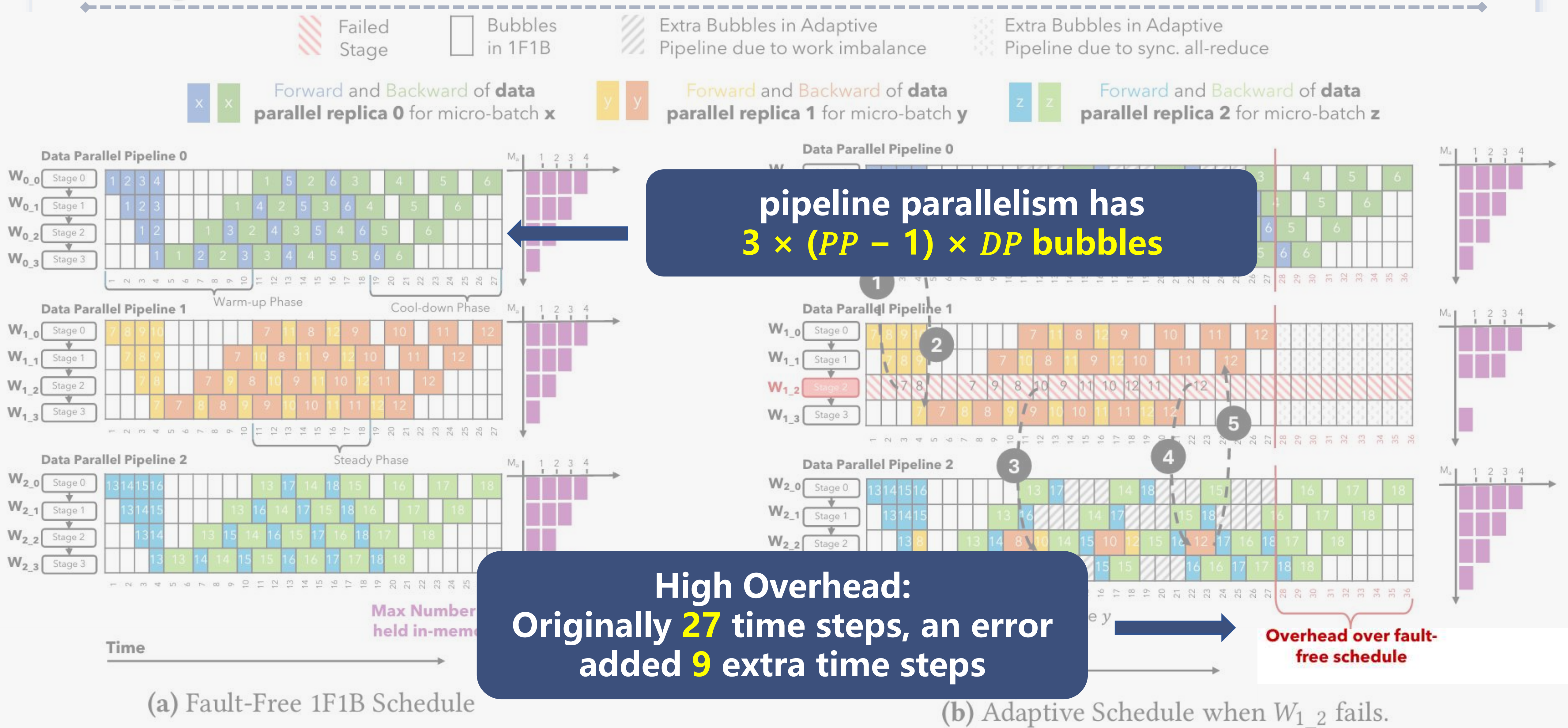
Design 1: Adaptive Pipelining



(a) Fault-Free 1F1B Schedule

(b) Adaptive Schedule when W_{1_2} fails.

Design 1: Adaptive Pipelining



pipeline parallelism has
3 × (PP − 1) × DP bubbles

High Overhead:
Originally 27 time steps, an error added 9 extra time steps

(a) Fault-Free 1F1B Schedule

(b) Adaptive Schedule when W_{1_2} fails.

Design： Efficient bubble filling

1. Decoupled BackProp: Filling Unused Bubbles
2. Staggered Optimizer: Accessing More Bubbles

Design 2: Decoupled BackProp

Backward

1. Backward computes input gradients and weights gradients
2. Layer i only depends on the input gradients from layer i+1
3. Weights gradients can be deferred until the end

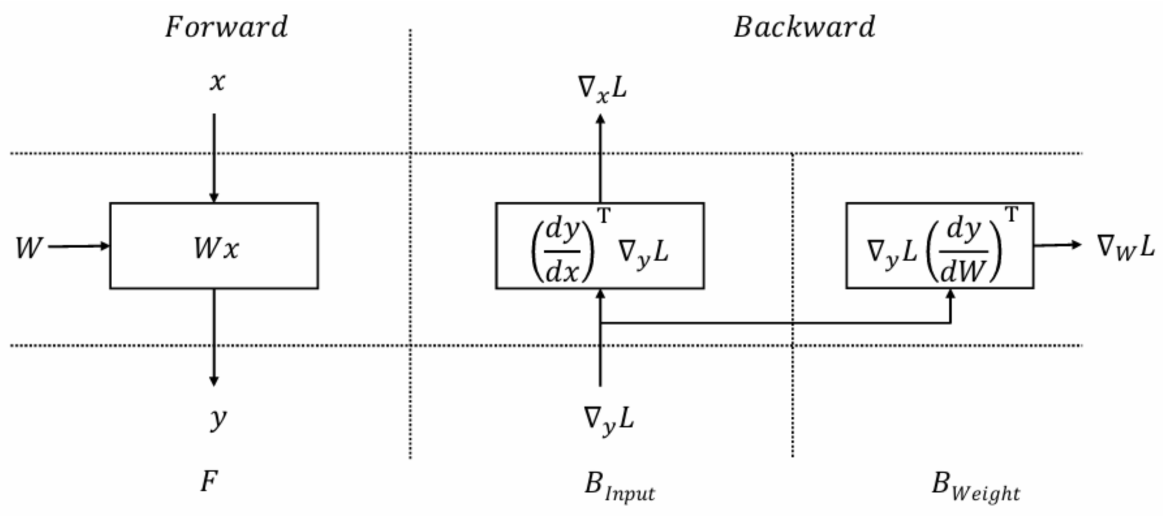


Figure 4. Forward and Backward pass for an operator.

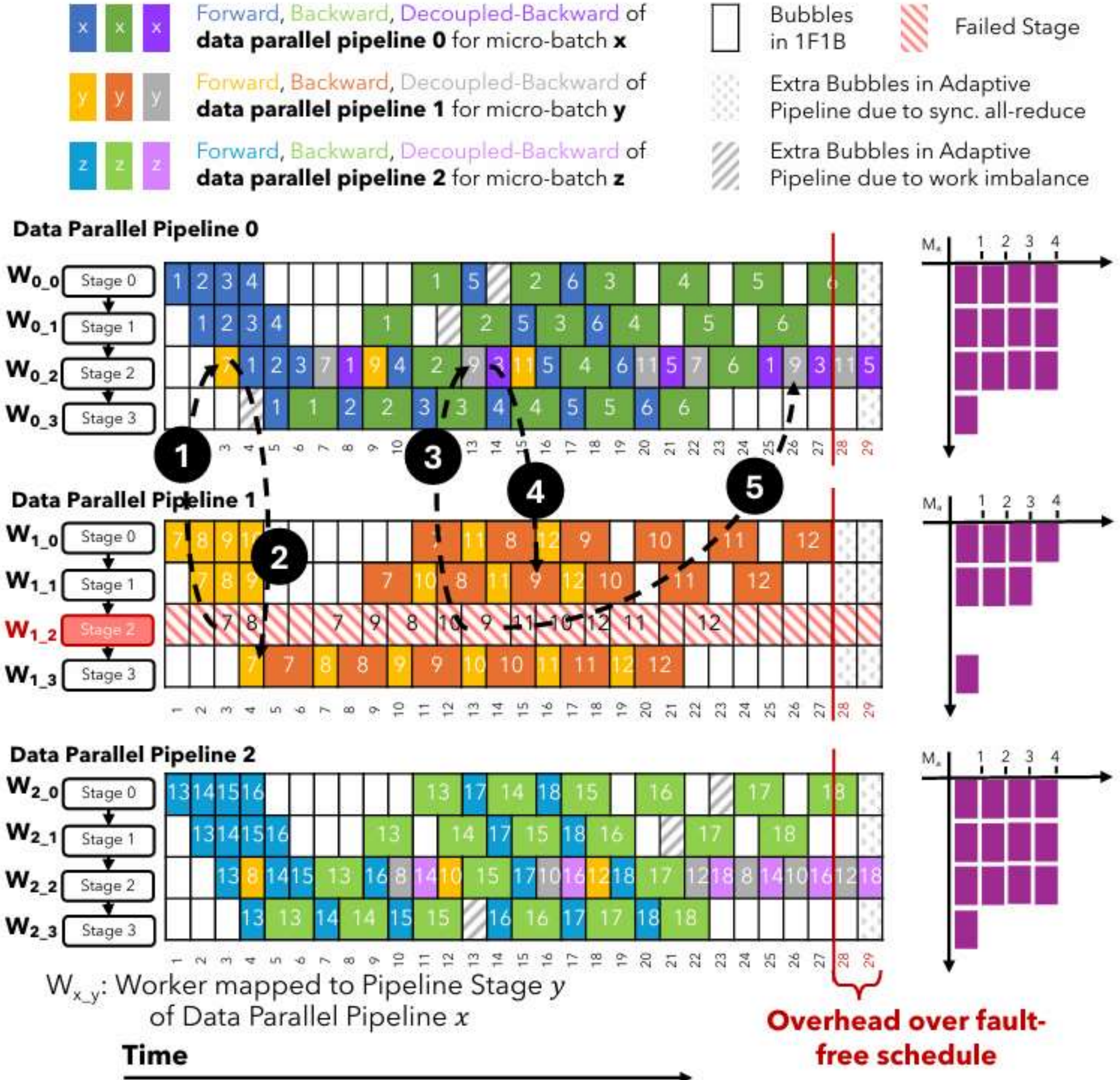
Design 2: Decoupled BackProp

Method

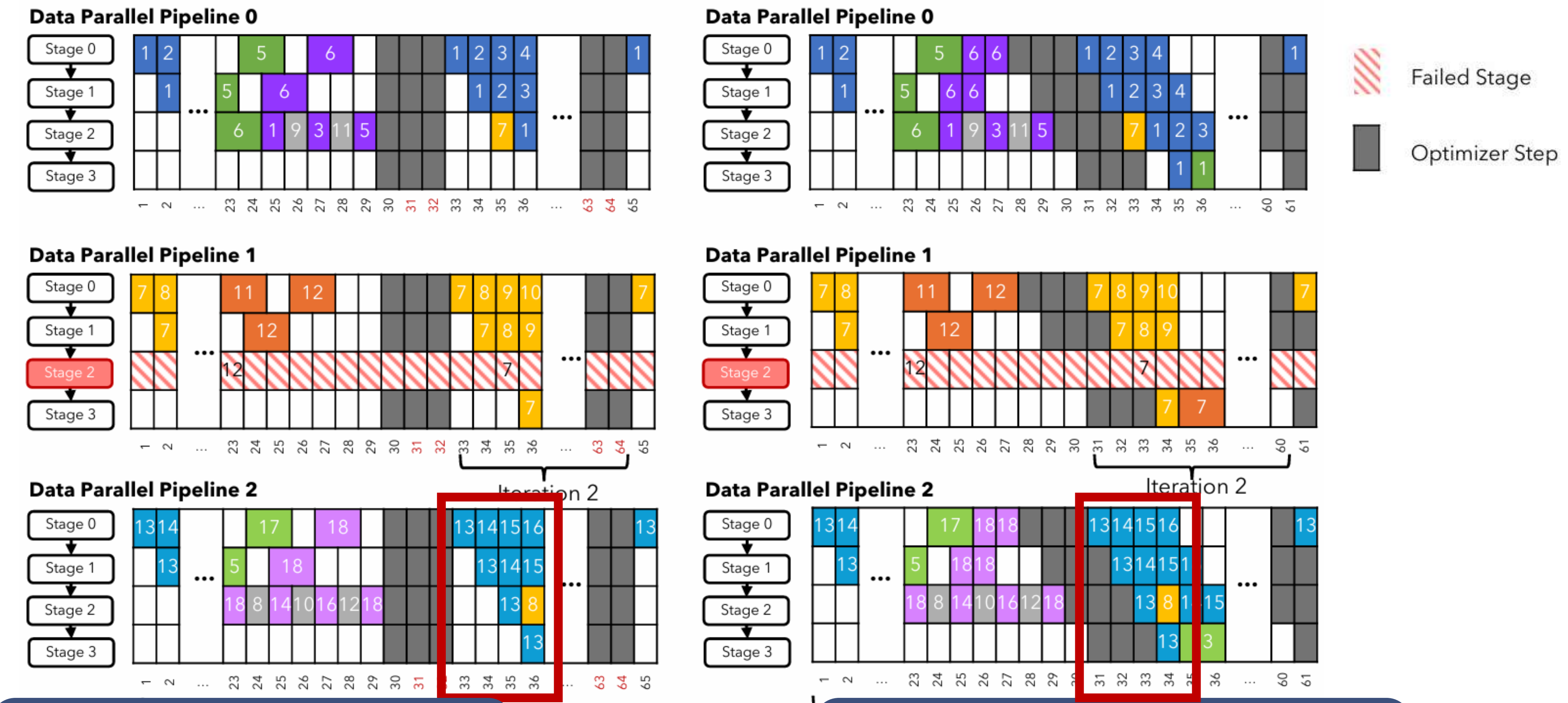
- Prioritize executing B_{Input} in bubbles
  - ✓ Advantage: extra time steps 9 -> 2
  - ✓ Disadvantage: Increases memory pressure

Memory pressure mitigation

- Avoid decoupling Backward unless necessary



Design 3： Staggered Optimizer

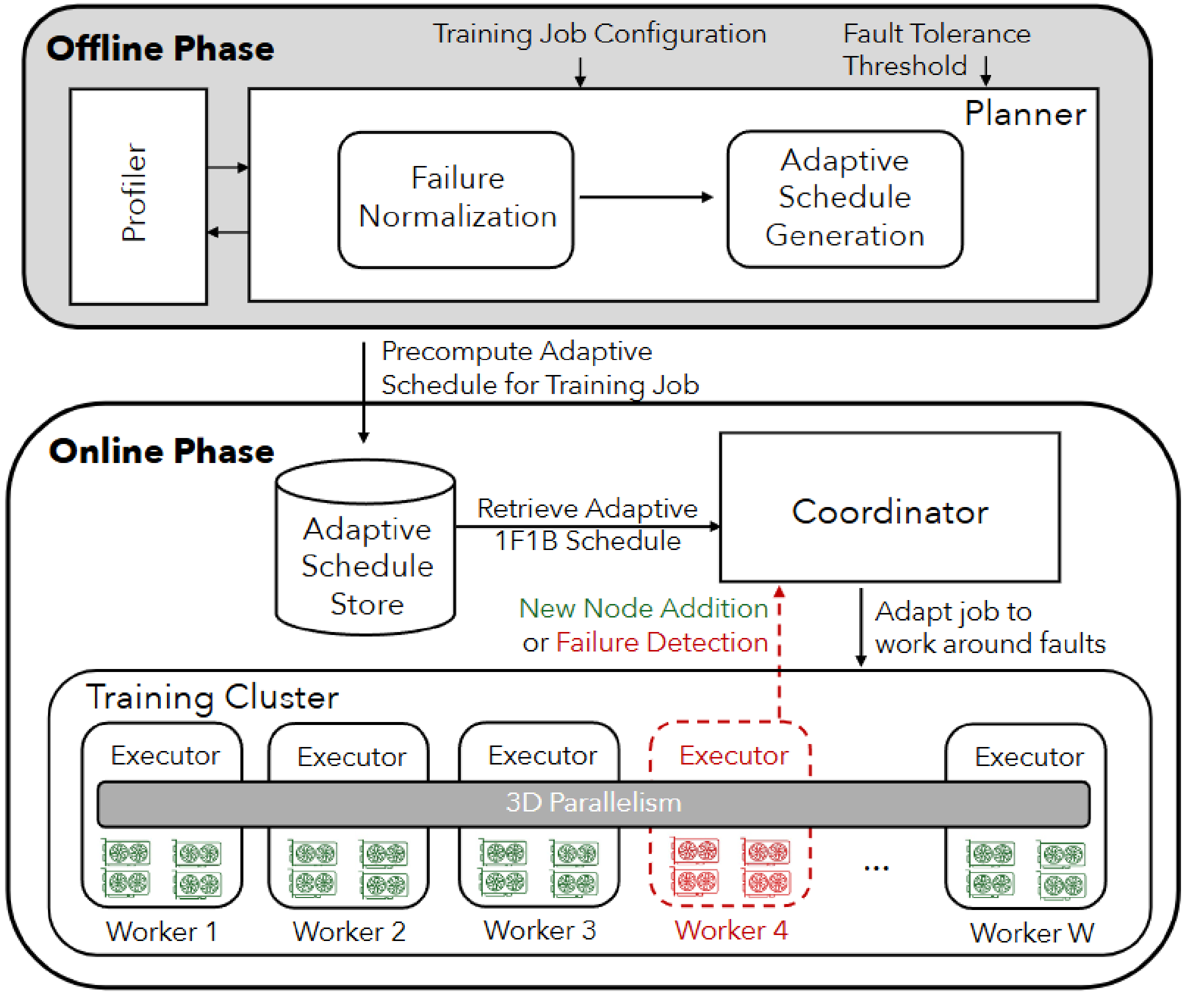


Synchronous optimizer -> warm-up bubbles

Asynchronous optimizer -> Reduce warm-up bubbles

System Overview

- System
  - Profiler
  - Planner
  - Executor
  - Coordinator



Planner: Failure Normalization

- Intuition
  - distribute across different peers
  - distribute to peers with more bubbles
- dynamic programming
  - F failures
  - PP pipeline stages
  - $A[i]$ = x, x failures in stage i
  - $O[i][f]$, handling f failures overhead

Algorithm 1 Failure Normalization

```
DP ← Number of data-parallel pipelines.
PP ← Number of pipeline stages.
MB ← Number of microbatches per pipeline.
F ← Total number of failures.
O ← an PP × (F+1) array                ▷ Rerouting Overheads
A ← an PP × (F+1) array                ▷ Assignments

procedure FAILURE_REORDERING(DP, PP, MB, F)
    for i ∈ {0, ..., PP − 1} do
        for f ∈ {0, ..., F} do
            if i == 0 then
                O[i][f] = COST(f)
                A[i][f] = [f]
            else
                x = arg min_{x≤f}(O[i − 1][f − x]
                        + COST(x))
                O[i][f] = O[i − 1][f − x] + COST(x)
                A[i][f] = concat(A[i − 1][f − x], x)
            end if
        end for
    end for
return A[PP − 1][F]
end procedure

procedure COST(f)
    if f > 0 then
        return min(0, MB×f×3−(DP−f)×(PP−1)×3)
    end if
    return 0
end procedure
```

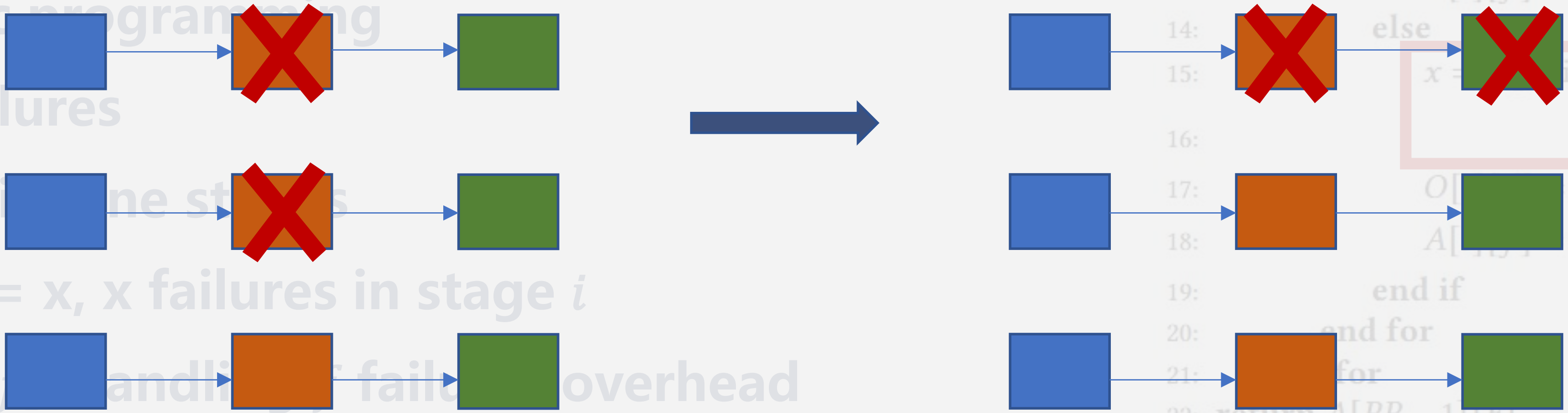
DP

additional time slots by heuristic

Planner: Failure Normalization

dynamic programming
Result: interrupted

dynamic programming
Result: uninterrupted



additional time slots by heuristic

Algorithm 1 Failure Normalization

$DP \leftarrow$ Number of data-parallel pipelines.
$PP \leftarrow$ Number of pipeline stages.
$MB \leftarrow$ Number of microbatches per pipeline.
$F \leftarrow$ Total number of failures.
$O \leftarrow$ an PP × (F+1) array ▷ Rerouting Overheads
$A \leftarrow$ an PP × (F+1) array ▷ Assignments

procedure $\text{COST}(f)$
  if $f > 0$ then
    return $\min\left(0, MB\times f\times 3-(DP-f)\times(PP-1)\times 3\right)$
  end if
  return 0
end procedure

Planner: Adaptive Schedule Generation

- MILP
  - T_{coom} : communication latency
  - T_F , T_{Binput} , $T_{Bweight}$: execution latency
  - operation (i, j, k, c, ks), $c \in \{F, Binput, Bweight\}$

a micro-batch ID 14, rerouted from $W2\_3$ to $W1\_3$: i = 3, j = 14, k = 2, ks = 1.

  - $S_{i,j,k}^{ks} \in \{0, 1\}$
  - $\sum_{ks} S_{i,j,k}^{ks} = 1$
  - $O_{(i,j,k,c,ks) \to (\grave{i},\grave{j},\grave{k},\grave{c},\grave{ks})} \in \{0, 1\}$
  - $E_{i,j,k,c}^{ks}$: ending time of operation (i, j, k, c, ks)

Planner: Adaptive Schedule Generation

- MILP
  - T_{coom}
  - T_F , T_{Binput} , $T_{Bweight}$
  - operation (i, j, k, c, ks)
  - $S^{ks}_{i,j,k} \in \{0, 1\}$
  - $\sum_{ks} S^{ks}_{i,j,k} = 1$
  - $O_{(i,j,k,c,ks)\to(\grave{i},\grave{j},\grave{k},\grave{c},\grave{ks})} \in \{0, 1\}$
  - $E^{ks}_{i,j,k,c}$

Cross-Stage Dependencies.

$$E^{k_s}_{i,j,k,F} \geq S^{k_s}_{i,j,k} \times (\sum_{\hat{k}} (E^{\hat{k}}_{i-1,j,k,F} \times S^{\hat{k}}_{i-1,j,k}) + T_{comm} + T_F) \quad (2)$$

$$E^{k_s}_{i,j,k,B_{Input}} \geq S^{k_s}_{i,j,k} \times (\sum_{\hat{k}} (E^{\hat{k}}_{i+1,j,k,B_{Input}} \times S^{\hat{k}}_{i+1,j,k}) + T_{comm} + T_{B_{Input}}) \quad (3)$$

Same-Stage Dependencies.

$$E^{k_s}_{i,j,k,B_{Weight}} \geq S^{k_s}_{i,j,k} \times (E^{k_s}_{i,j,k,B_{Input}} + T_{B_{Weight}}) \quad (4)$$

No Overlapping Computations.

$$E^{k'_s}_{i,j',k',c'} \geq E^{k'_s}_{i,j,k,c} + T_{c'} - \infty(1 - S^{k'_s}_{i,j,k} \times S^{k'_s}_{i,j',k'} + O_{(i,j,k,c,k'_s)\to(i,j',k',c',k'_s)}) \quad (5)$$

Planner: Adaptive Schedule Generation

MILP

- A_B , $A_{Bweight}$: activation
- A_{Binput} , $A_{Bweight}$: gradients
- operation (i, j, k, c, ks)
- $O_{(i,j,k,c,ks)\rightarrow(\grave{i},\grave{j},\grave{k},\grave{c},\grave{ks})} \in \{0, 1\}$

$$\Delta M^{k_s}_{i,j,k,c} = \begin{cases} A_B & \text{, if } c = F \text{ and } S^{k_s}_{i,j,k} = 1 \\ A_B - A_{B_{Input}} & \text{, if } c = B_{Input} \text{ and } S^{k_s}_{i,j,k} = 1 \\ -A_{B_{Weight}} & \text{, if } c = B_{Weight} \text{ and } S^{k_s}_{i,j,k} = 1 \\ 0 & \text{, otherwise} \end{cases}$$

Memory Constraint.

$$M_{Limit} \geq \underline{\Delta M^{k'_s}_{i,j',k',c'}} + \sum_{j,k,c} \underline{\Delta M^{k'_s}_{i,j,k,c}} \times O_{(i,j,k,c,k'_s)\rightarrow(i,j',k',c',k'_s)} \tag{6}$$

Implementation on DeepSpeed

- Rerouting: communication operators
  - ReRouteAct
  - ReRouteGrad
- Decoupling BackProp: pipeline instructions
  - InputBackwardPass
  - WeightBackwardPass
- Rerouting: communication operators
  - optimizer in pipeline stage

Experimental Setup

- Cluster Setup
  - 4 Standard_NC96ads_A100_v4 (8 A100 GPUs, 96 vCPUs, and 880 GB memory each) in Azure, 600 GB/s NVLink intra-node, 640 Gbps internode
- Baselines
  - Bamboo, Oobleck
- Workloads
  - GPT-3: Medium (350M), 3.35B, and 6.7B
  - (PP, DP): (2, 16), (4, 8), and (8, 4)
  - WikiText
  - Train 6 hours

Training Throughput Under Failures

- Bamboo: redundant computations and additional model state copies
- Oobleck: imbalanced pipelines and higher reconfiguration latency (re-shuffle)

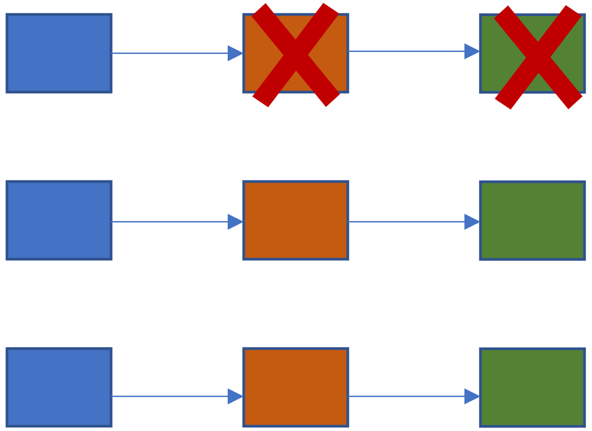


Table 1. Training throughput (samples/sec) with increasing failure frequency, higher is better. Bamboo ran out of memory for GPT-3 3.35B and 6.7B.

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

Training Throughput Under Failures

- 1.64× improvement over Bamboo，1.46× improvement over Oobleck

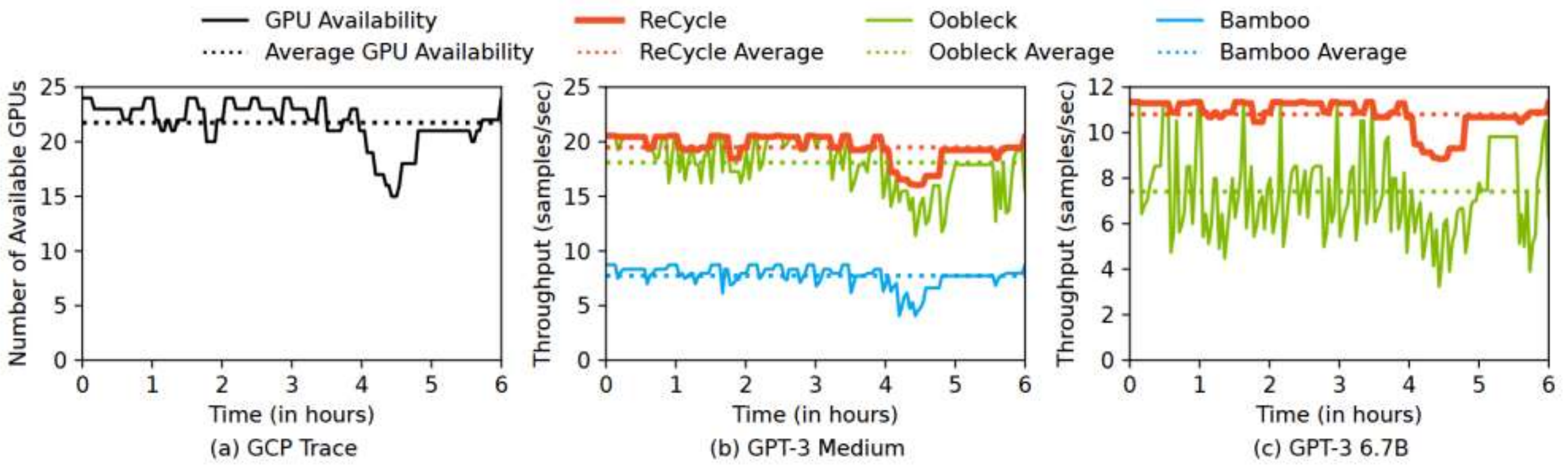


Figure 9. Training throughput (samples/sec), higher is better, for the GPT-3 Medium and GPT-3 6.7B models over the GCP trace. In 9b and 9c, the dashed lines represent the average training throughput achieved by each system within the 6h period.

ReCycle Scalability

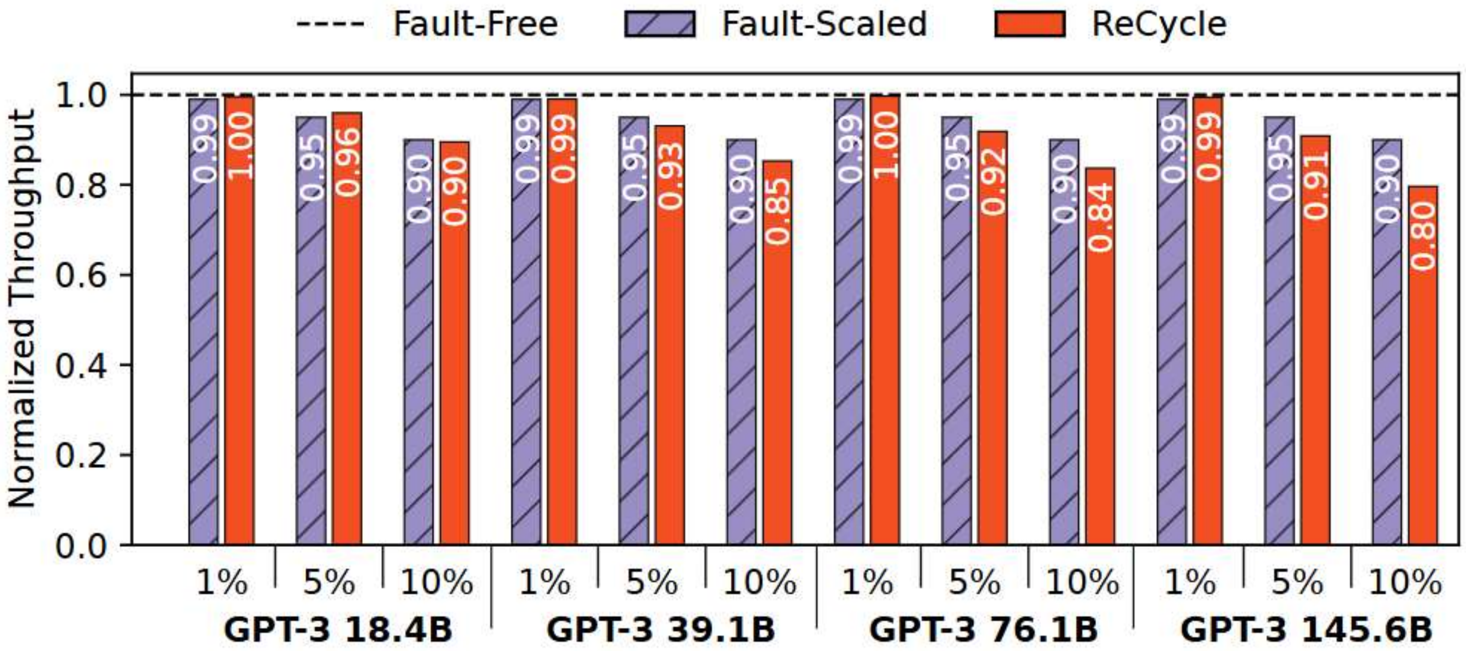
- Simulator
  - simulate maximum discrepancy is 5.98%
  - variations from minor fluctuations by NCCL collectives

Table 2. Gap between real-world and simulated throughput across various models and failure rates.

Models	Fault-Free	6h	2h	30m
GPT-3 Medium	-0.87%	+5.98%	-1.93%	-1.48%
GPT-3 3.35B	-0.13%	-1.58%	+2.12%	-1.90%
GPT-3 6.7B	+3.94%	+2.71%	-1.86%	-0.85%

ReCycle Scalability

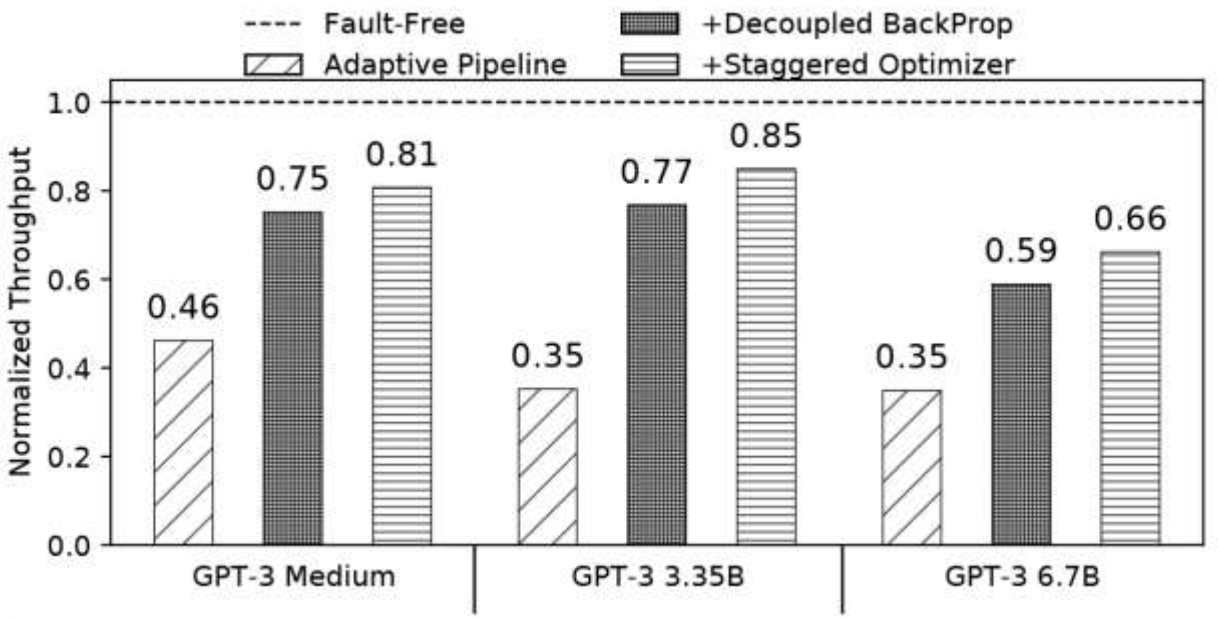
- Large scale simulation
  - At a failure rate of 1%-5%,the performance of ReCycle is comparable to that of Fault-Scaled
  - At a failure rate of 10%, the performance of ReCycle degrades by 0.5% to 11.5%



256 GPUs 512 GPUs 1024 GPUs 1536 GPUs

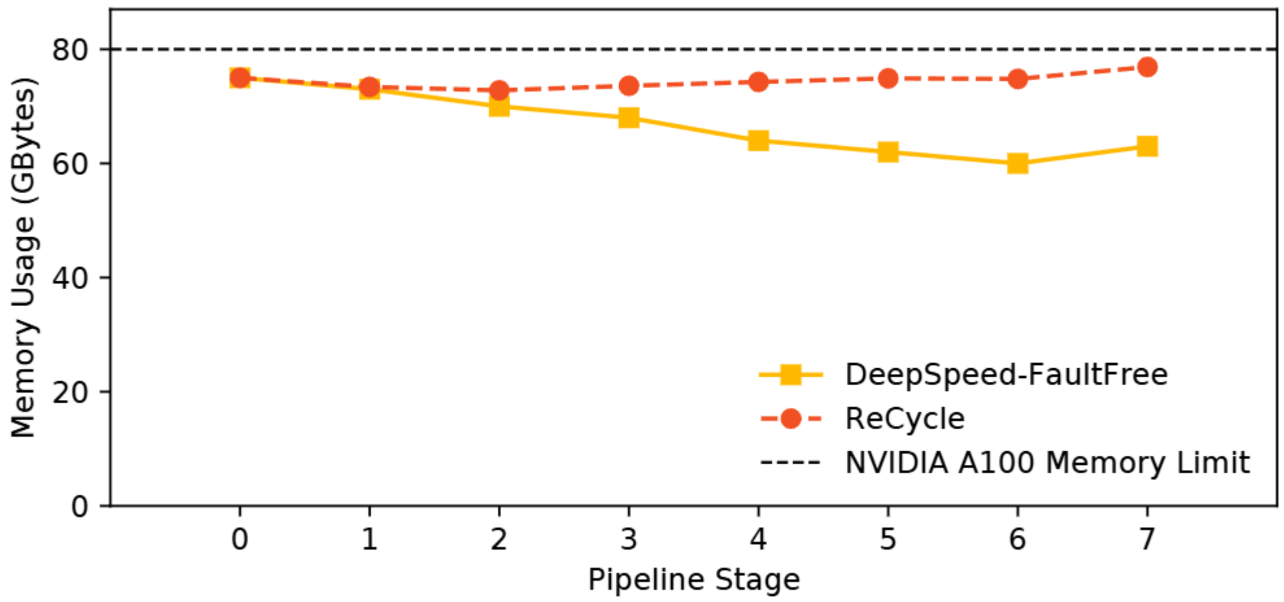
ReCycle Performance Breakdown

- Adaptive Pipelining: additional work
- Decoupled BackProp: effectively utilizing bubble, improve 63% to 118%
- Staggered Optimizer: reduce warm-up bubbles, improve 7% to 11%



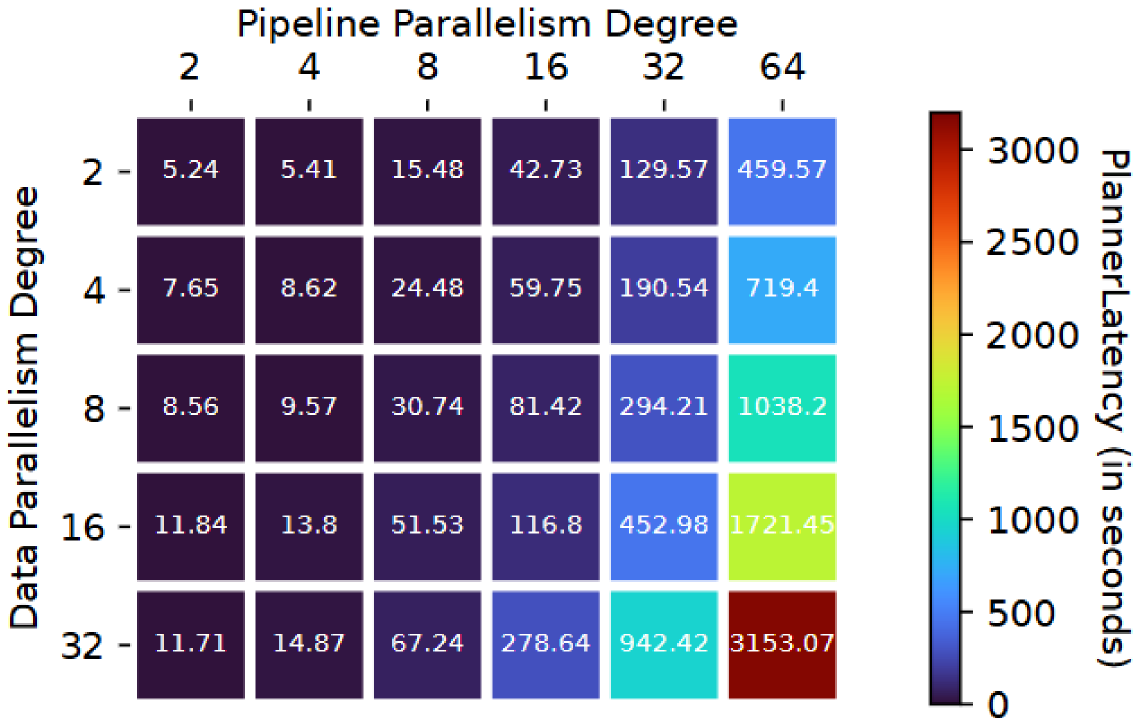
ReCycle Performance Breakdown

- **By decoupling BackProp and delaying B_{weight} computation, nearly full utilization of GPU memory is achieved**



Planner Overhead

- **With 25% GPU failure, the Planner finds optimal scheduling with a delay of less than 0.1% of the total training time**



Conclusion

- Pros:
  - Technical Advantages: Continuous training using data parallelism
  - Technical Advantages: Combined optimization of bubbles
  - Paper Advantages: The images clearly express the core design
- Cons:
  - Dynamic programming will probably interrupt training
  - Fine-grained scheduling of bubbles is difficult, and the paper does not explain how to achieve it
  - Existing pipeline parallelism techniques have already optimized the bubbles