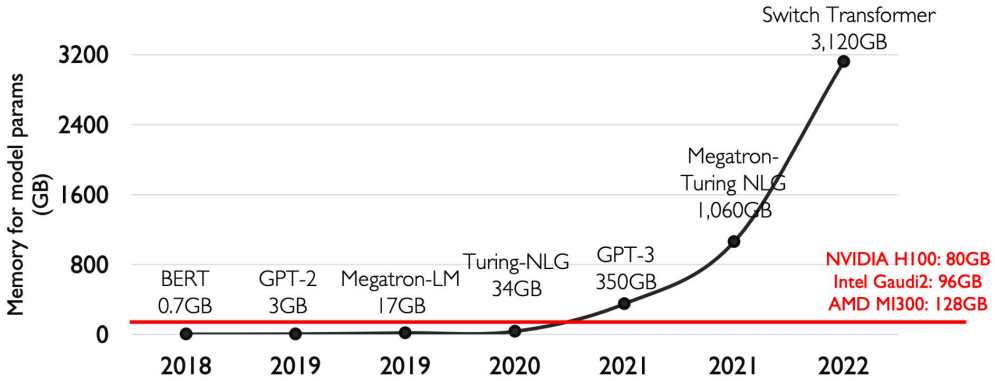
CSE 585 Presentation September 17, 2024

Yi Chen, Yunchi Lu, Kevin Xue

Exponentially Increasing Model Size



From [Jang et al, SOSP '23 Technical Sessions]

(Assuming 2 bytes per parameter)

Models do not fit in a single GPU

(Actually, we need thousands of them)

Rescue: Parallelism,
But
Efficient & Reliable
Parallelism

Introduction

Efficient & Reliable

Combines different parallelism strategies to make model training efficient

Uses pipeline templates to make model training reliable

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

Deepak Narayanan‡*, Mohammad Shoeybi†, Jared Casper†, Patrick LeGresley†,
Mostofa Patwary†, Vijay Korthikanti†, Dmitri Vainbrand†, Prethvi Kashinkunti†,
Julie Bernauer†, Bryan Catanzaro†, Amar Phanishayee*, Matei Zaharia‡
†*NVIDIA* ‡*Stanford University* **Microsoft Research*

Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates

Insu Jang
insujang@umich.edu
University of Michigan

Zhenning Yang
znyang@umich.edu
University of Michigan

Zhen Zhang
zhzhn@amazon.com
Amazon Web Services

Xin Jin
xinjinpku@pku.edu.cn
Peking University

Mosharaf Chowdhury
mosharaf@umich.edu
University of Michigan

Efficient Large-Scale Language Model Training on GPU Clusters Using

Megatron-LM

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, Matei Zaharia

This Paper

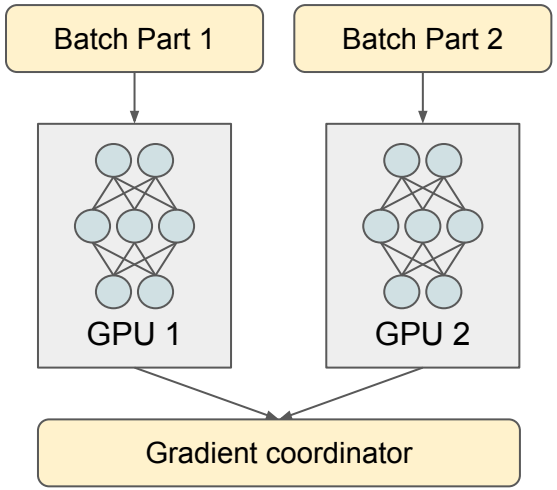
- What existing parallelism strategies are currently used?

- An analytical model for the tradeoffs among these strategies
  - Goal: combining different techniques for optimal training efficiency

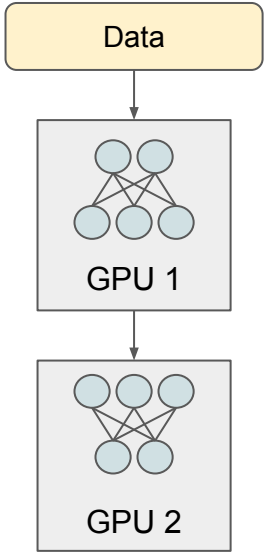
- How well does the analytical model perform in real-world training?

Parallelism Strategies

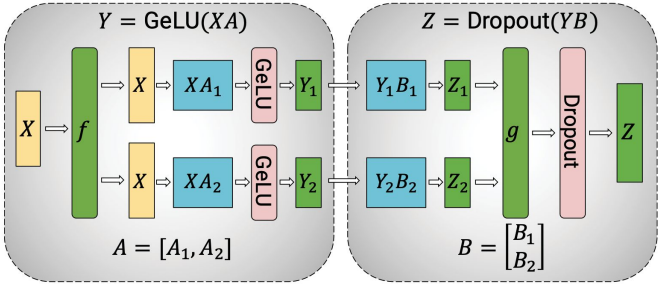
Data Parallelism (DP)



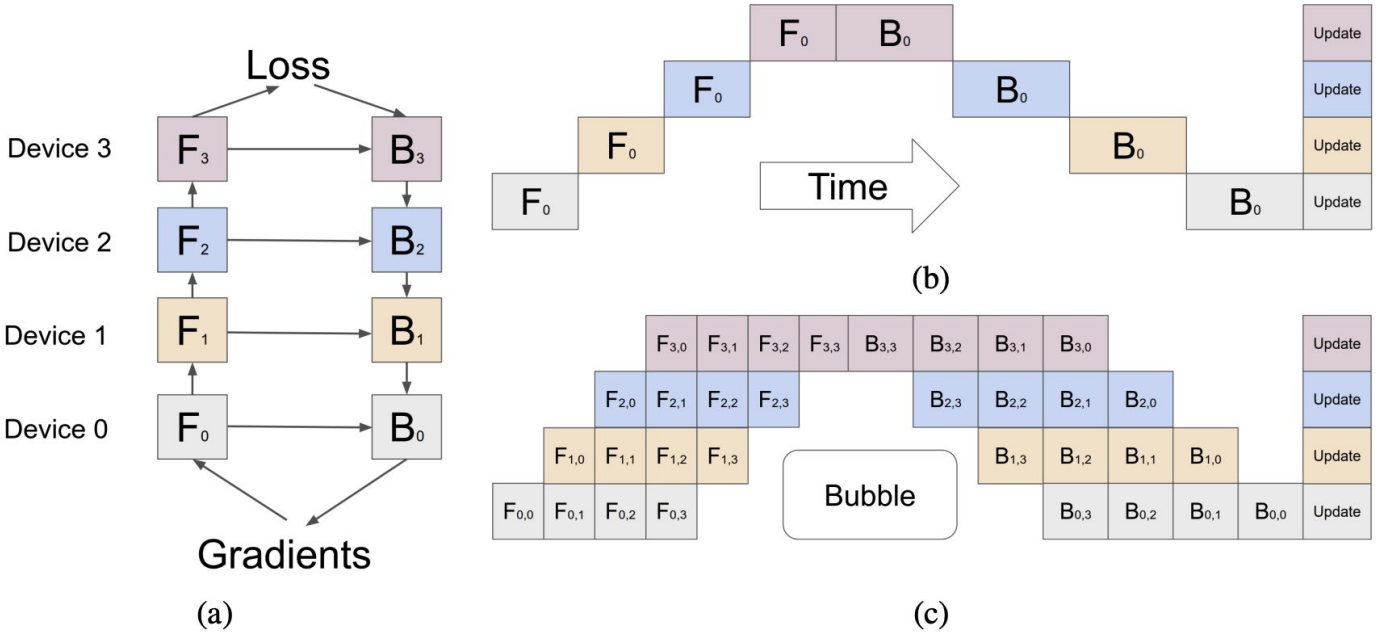
Pipeline (Model) Parallelism (PP)



Tensor (Model) Parallelism (TP)



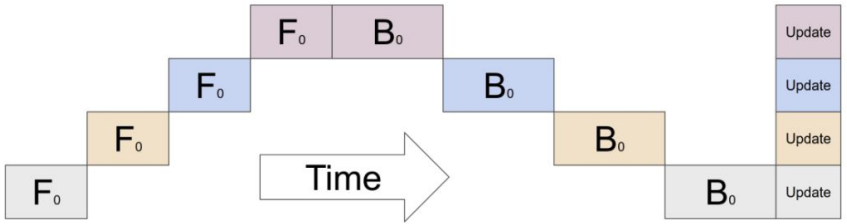
Pipeline Parallelism



From [Huang et al, GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism (NeurIPS 2019)]

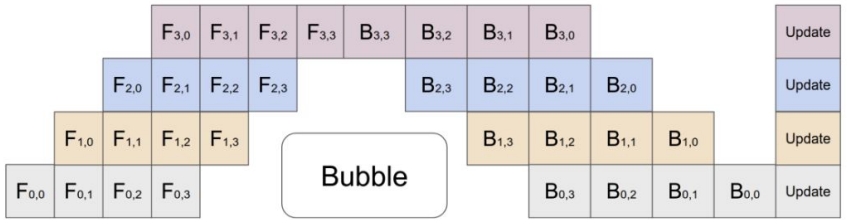
Pipeline Parallelism - Optimization

# microbatches
m = 1



(b)

# microbatches
m = 4



(c)

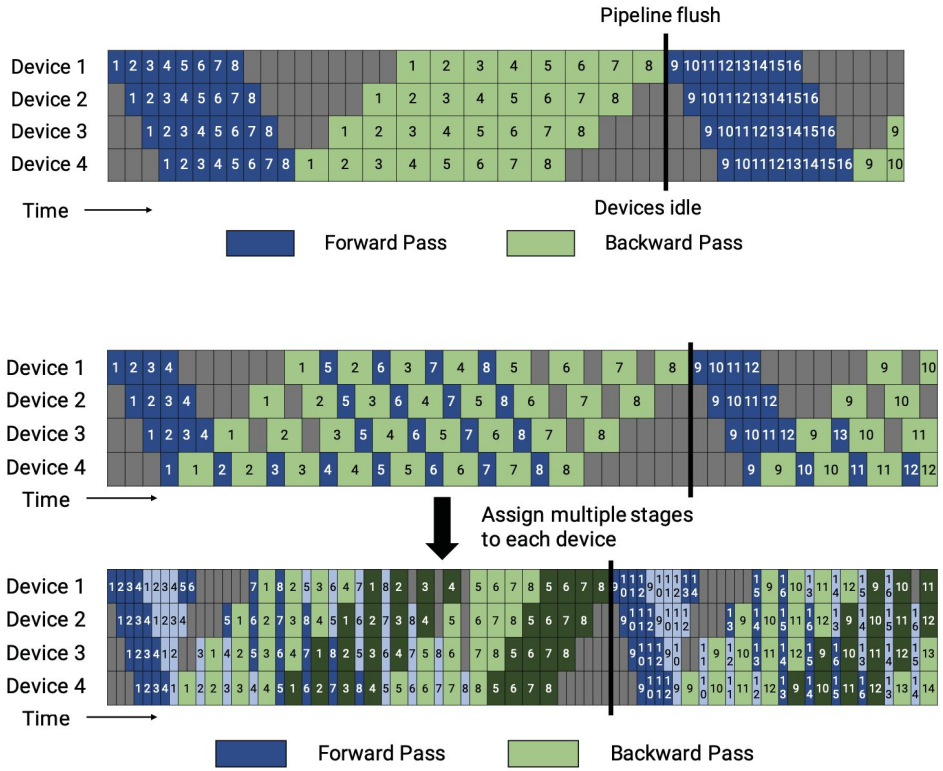
Takeaway:
More microbatches → Higher pipeline utilization

From Huang et al, GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism (NeurIPS 2019)

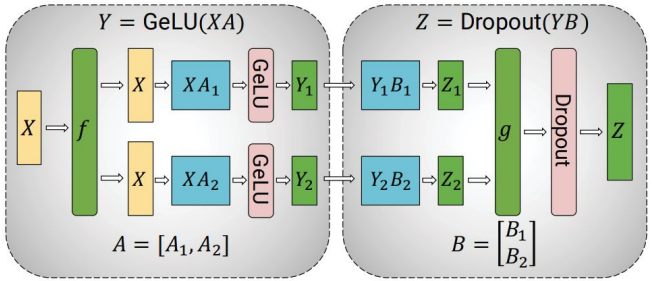
Pipeline Parallelism - Optimization (Cont'd)

Optimize for memory usage

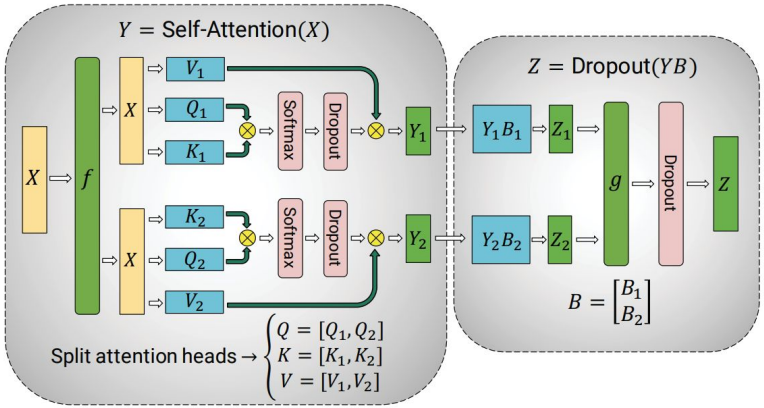
Optimize for pipeline time



Tensor Parallelism



(a) MLP.



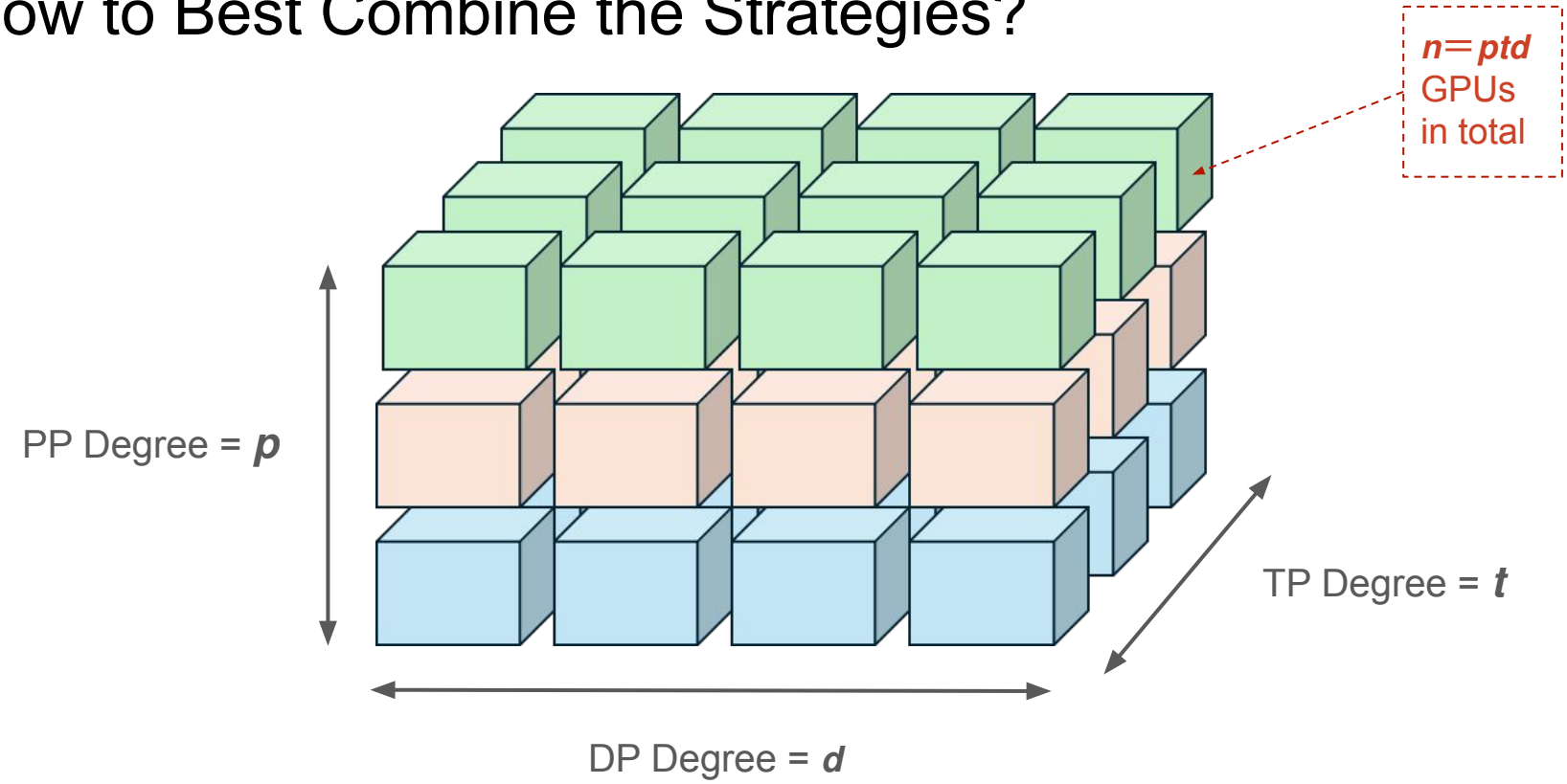
(b) Self-Attention.

- Split the weight matrix
- Synchronize with all-reduce
- ***f*(.)**:
  - identity mapping forward
  - all-reduce backward
- ***g*(.)**:
  - all-reduce forward;
  - identity mapping backward

Comparison of Parallelism Strategies

Data Parallelism (DP)	Pipeline Parallelism (PP)	Tensor Parallelism (TP)
+ High training speed (batches processed in parallel) + Lower memory footprint for activation - Cannot fit large model to a single GPU	+ Can split the large model into multiple GPUs + Point-to-point communication - Underutilized pipelines	+ Can split the large model into multiple GPUs + Good utilization and work fairness - Large communication overhead due to all-reduce

How to Best Combine the Strategies?



Model Parallelism (TP vs PP)

Perspective 1: reducing bubble size

- Assuming $n = ptd$ is fixed
- If t increases, then p will decrease, and the bubble will become smaller

Premise 1: more tensor parallelism → better pipeline utilization

Perspective 2: reducing communication

- All-reduce is expensive

Premise 2: avoid tensor parallelism across nodes

Model Parallelism - Takeaway

Combining these two premises:

- Premise 1: more TP → better pipeline utilization
- Premise 2: avoid TP across nodes

Takeaway: use TP in GPUs within a compute node, but PP across the compute nodes

Data vs Model Parallelism

Data parallelism vs tensor parallelism

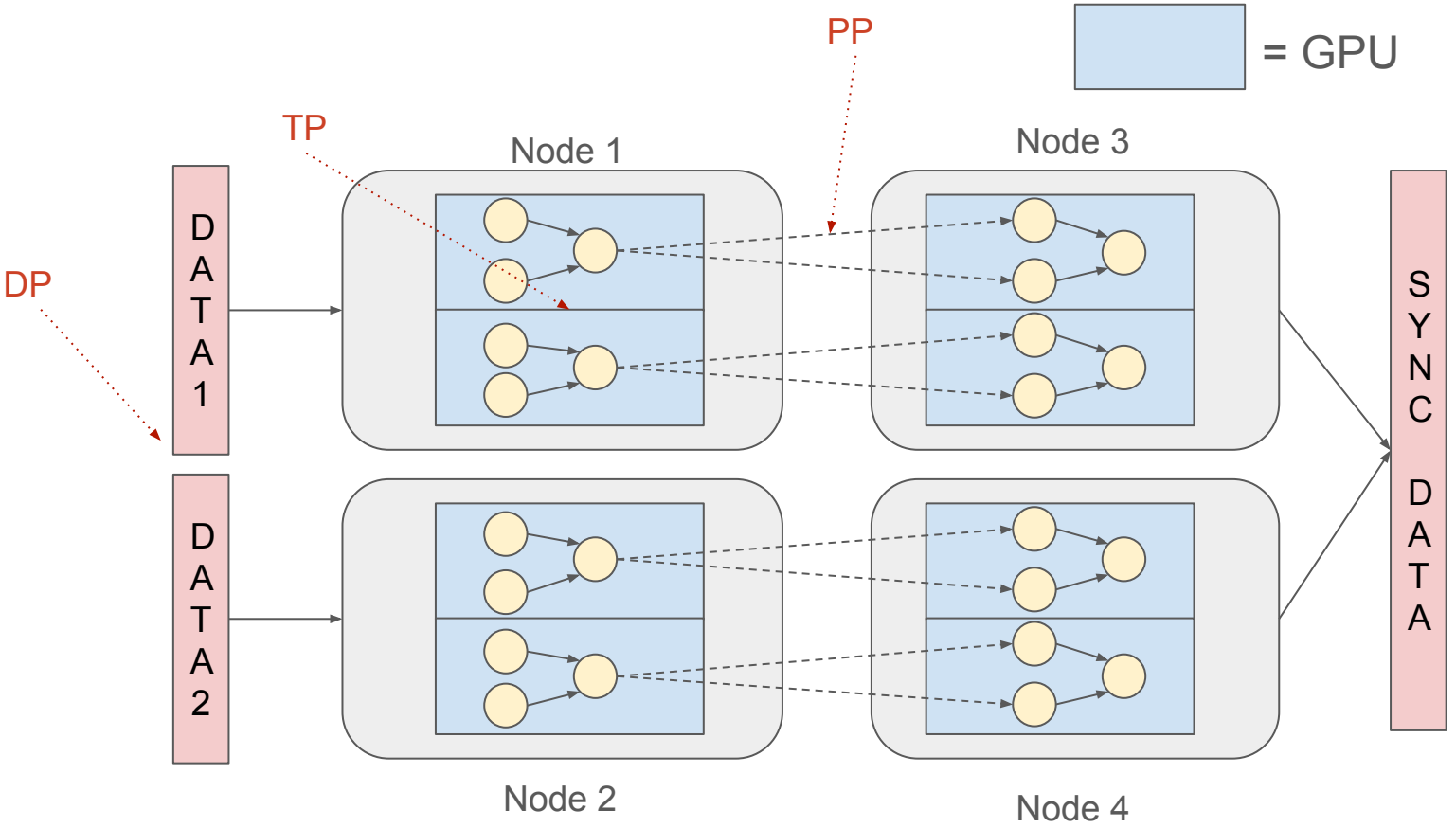
- TP incurs more communication overhead per microbatch; DP's all-reduce is only per-batch

Data parallelism vs pipeline parallelism

- Same argument as TP vs PP: *more DP* → less PP → *more utilization*

Takeaway: prefer DP over TP and PP so long as the model fits in single GPU memory

PTD-P: Pipeline, Tensor, and Data Parallelism

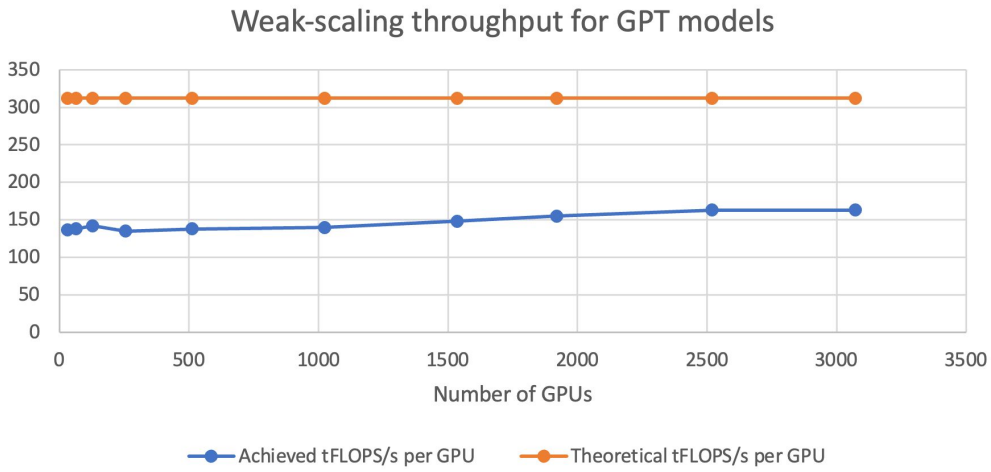


Evaluation

- Platform
  - Each node in cluster has
    - 8 * NVIDIA 80GB A100 with NVLink
    - 10 * NVIDIA Mellanox 200Gbps HDR Infiniband HCA
  - Three-level fat-tree topology (better for all-reduce)
- Workload
  - GPT models of appropriate sizes
- Questions to be answered
  - How well does PTD-P perform
  - How do different parallelization dimensions interact with each other

Evaluation - How well does PTD-P perform?

- Absolute throughput
  - ~50% of GPU's capability. Still have room for improvement
- Scaling
  - Pretty good



Evaluation - How well does PTD-P perform?

- Compared with ZeRO-3 (SOTA)
  - Better throughput
  - Better scaling
- Compared with theoretical peak throughput (312 FLOPS/s)
  - ~50%. Still have room for improvement

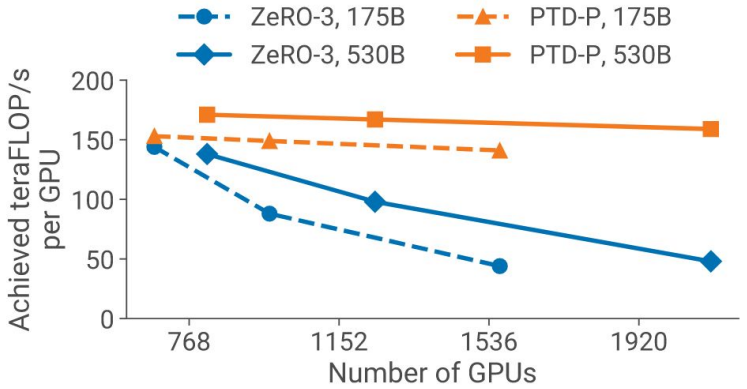


Figure 10: Throughput per GPU of PTD-P and ZeRO-3 for two different GPT models (the 175B GPT-3 model is shown with dotted lines, and the 530B model is shown with solid lines). Global batch sizes are fixed and ZeRO-3 is used without any model parallelism.

Evaluation - Comparison of Parallel Configurations

- TP - PP
  - Matches the conclusion that: **use TP in GPUs within a compute node, but PP across the compute nodes**

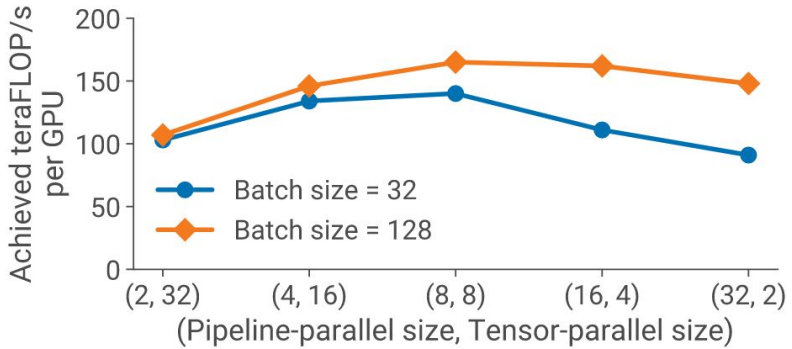


Figure 13: Throughput per GPU of various parallel configurations that combine pipeline and tensor model parallelism using a GPT model with 162.2 billion parameters and 64 A100 GPUs.

Evaluation - Comparison of Parallel Configurations

- TP/PP - DP
  - Matches the conclusion that: **prefer DP over TP and PP so long as the model fits in single GPU memory**

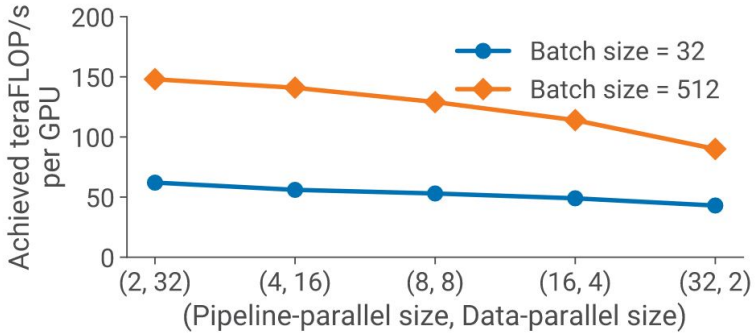


Figure 14: Throughput per GPU of various parallel configurations that combine data and pipeline model parallelism using a GPT model with 5.9 billion parameters, three different batch sizes, microbatch size of 1, and 64 A100 GPUs.

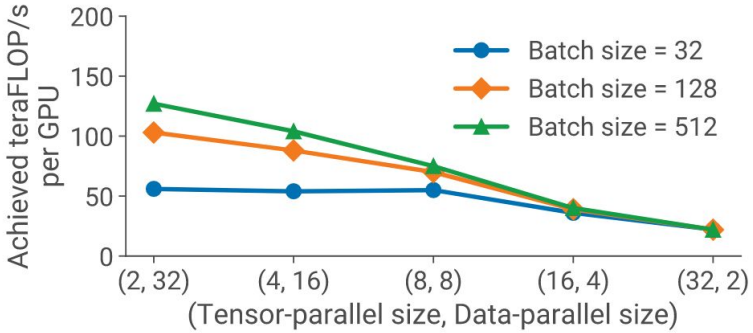


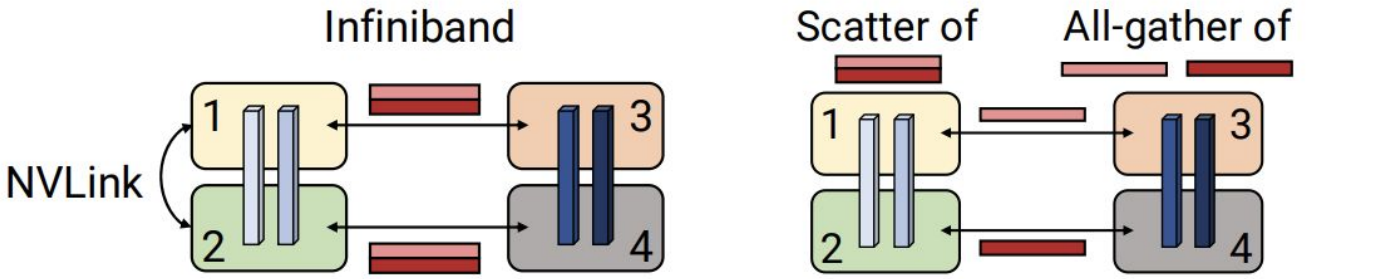
Figure 15: Throughput per GPU of various parallel configurations that combine data and tensor model parallelism using a GPT model with 5.9 billion parameters, three different batch sizes, microbatch size of 1, and 64 A100 GPUs.

Megatron-LM Backup 1: Notation

Notations:

- p, t, d: parallelism dimensions
- $n = ptd$: number of GPUs
- B: global batch size
- $m = B/bd$ microbatch size

Megatron-LM Backup 2: Optimization for Communication



(a) W/o scatter/gather optimization. **(b) With scatter/gather optimization.**

Benefits

- Better use of the point-to-point send/receive of the InfiniBand cards
- Avoids redundantly sending replica of the all-reduced activation

Megatron-LM Backup 3:Activation Recomputation

- Running the forward pass a second time just before the backward pass to keep memory footprint acceptably low

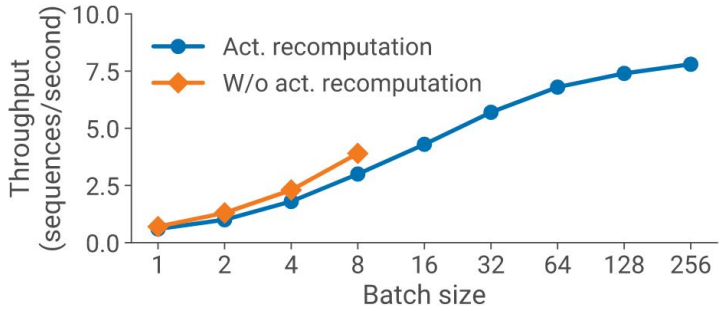


Figure 17: Throughput (in sequences per second) with and without activation recomputation for a GPT model with 145 billion parameters using 128 A100 GPUs ($(t, p) = (8, 16)$).

Megatron-LM Backup 4: Pros & Cons

Pros

- Impressive performance
  - Throughput
  - Scalability
- An analytical model about choosing the best strategy

Cons

- Manual parallel plan tuning

Oobleck

Resilient Distributed Training of Large Models Using Pipeline Templates

Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury

credit to Insu Jang for sharing his slides

DNN models continue to become larger

- As the figure we showed when presenting Megatron-LM
  - hundreds of billions of **parameters**
  - E.g. Switch Transformer - 39 * 80GB GPUs to store the model

→ **Distributed hybrid-parallel training**

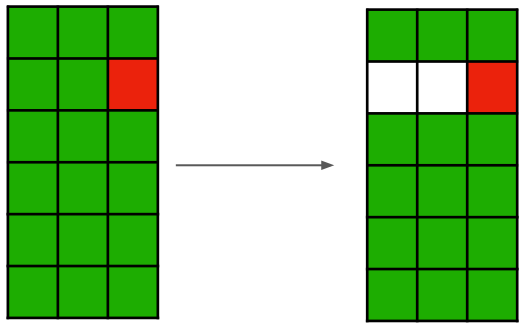
Failures Getting Noticeable

- More GPU are used
  - Higher failure rate
  - Higher failure cost

- For example, during a **two month** distributed training for the OPT model, Meta reported over 105 training restarts resulting from failures on more than 100 VMs in the cloud, indicating **1.25 incidents per day** and **61k GPU hours** affected (Susan Zhang et al. {OPT}, arXiv:2205.01068)

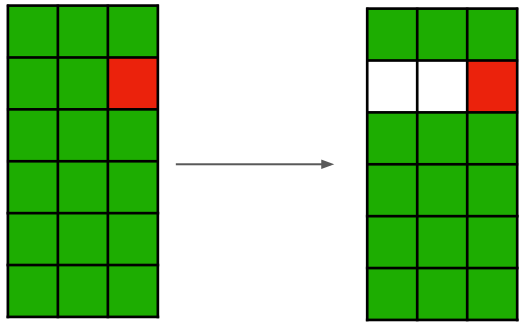
Existent Solutions - **Checkpointing**

- Traditionally, requires manual reconfiguration after identifying and replacing the failed GPUs with spare ones, which is time-consuming
- **Varuna**: dynamic reconfiguration, recovery automation



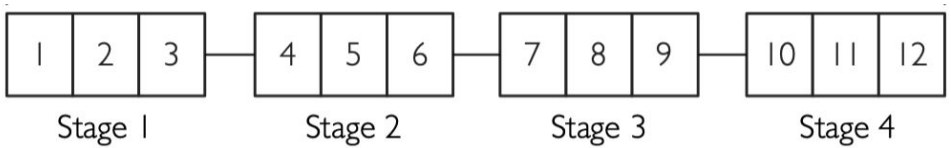
Existent Solutions - **Checkpointing**

- Traditionally, requires manual reconfiguration after identifying and replacing the failed GPUs with spare ones, which is time-consuming
- **Varuna**: dynamic reconfiguration, recovery automation
  - Non-proportional impact
  - High recovery time



Existent Solutions - **Redundant Computation**

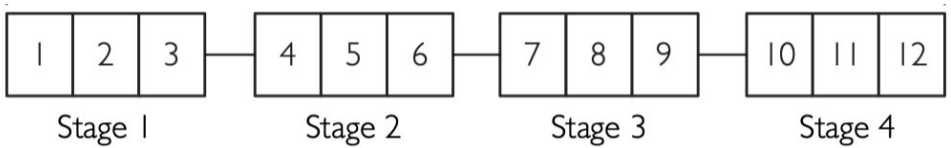
- To avoid reconfiguration and restart overheads, **Bamboo** introduced redundant computation



Node 1: Stage 1, 2
Node 2: Stage 2, 3
Node 3: Stage 3, 4
Node 4: Stage 4, (1)

Existent Solutions - **Redundant Computation**

- To avoid reconfiguration and restart overheads, **Bamboo** introduced redundant computation

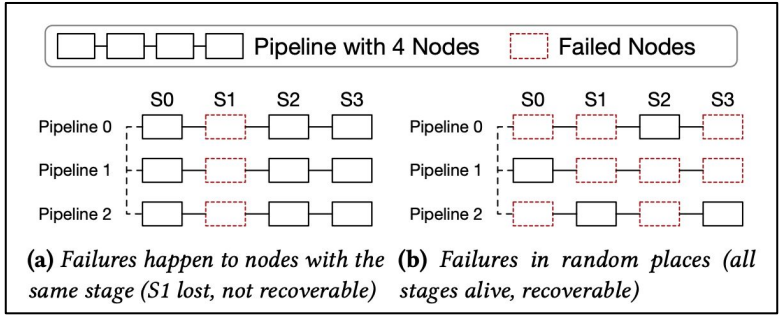


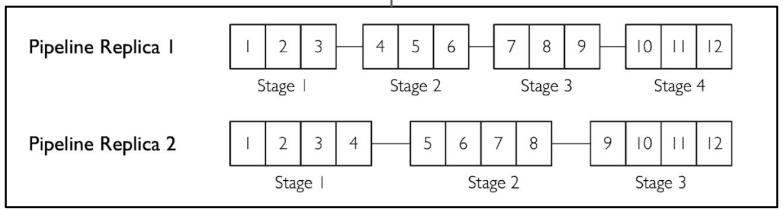
Node 1: Stage 1, 2
Node 2: Stage 2, 3
Node 3: Stage 3, 4
Node 4: Stage 4, (1)

- Significant overhead
- Unsatisfying failure tolerance

Oobleck Overview

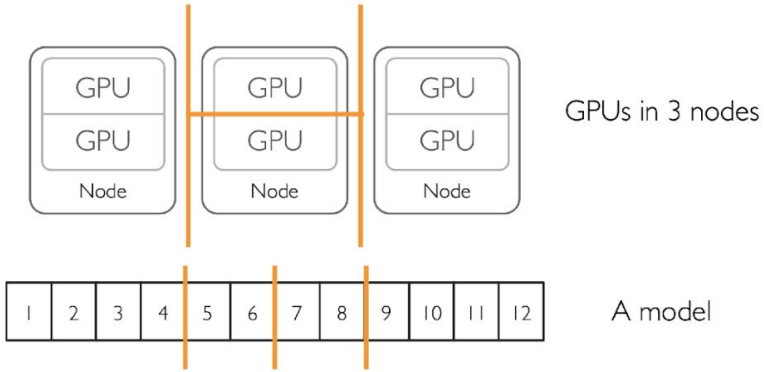
- Provide **guaranteed** fault tolerance
- Utilize **inherent redundancy** in data parallel
- Achieve fast recovery by eliminating checkpointing
- Introducing **pipeline template**

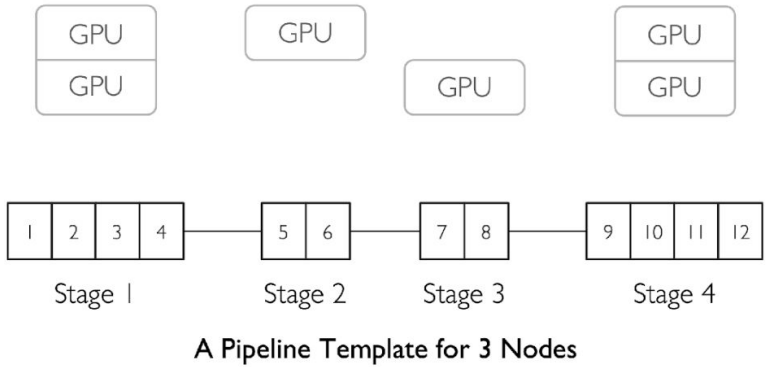




Pipeline Template

- Each template is a pre-generated single pipeline execution specification for specific number of nodes

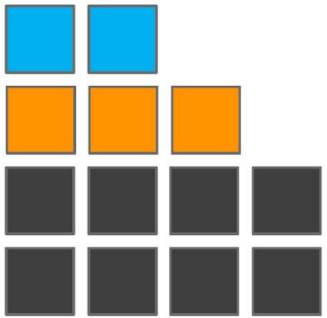




A Pipeline Template for 3 Nodes

Parallel Execution Plan

- Parallel execution plan is configured as **a linear combination of templates**

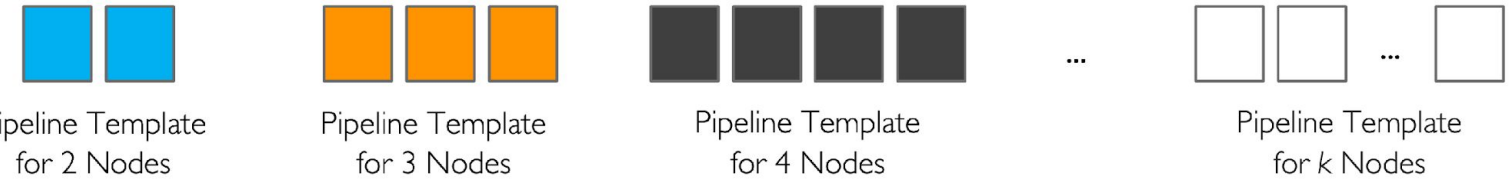


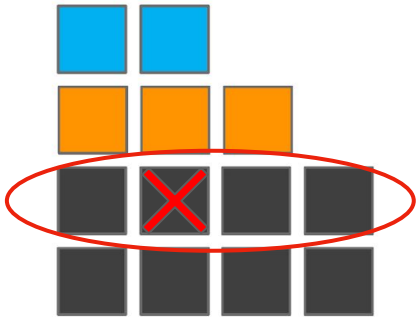
An example of parallel execution plan with 13 nodes

1x pipeline from the pipeline template for 2 nodes
1x pipeline from the pipeline template for 3 nodes
2x pipelines from the pipeline template for 4 nodes

Parallel Execution Plan

- Parallel execution plan is configured as **a linear combination of templates**



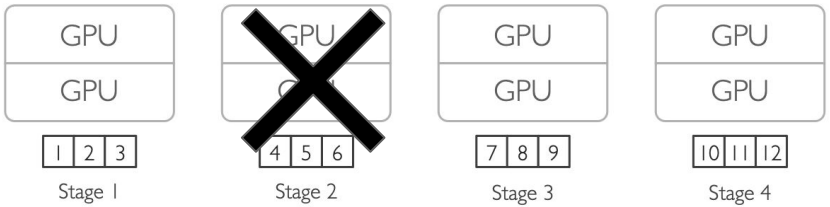


An example of parallel execution plan with 13 nodes

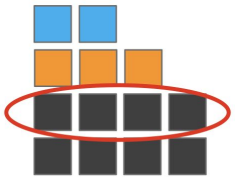
1x pipeline from the pipeline template for 2 nodes
1x pipeline from the pipeline template for 3 nodes
2x pipelines from the pipeline template for 4 nodes

Parallel Execution Plan

- Quickly reinstantiate a new pipeline from a template when failures happen

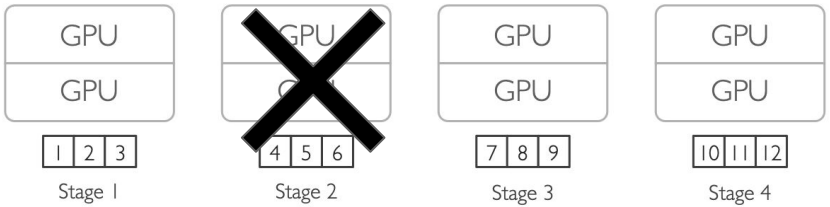


A pipeline instantiated from the template for 4 nodes

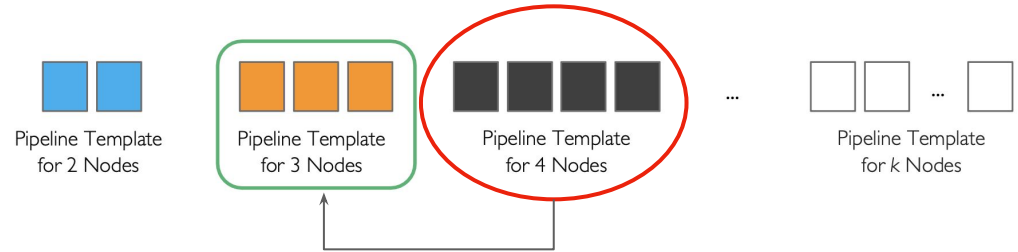


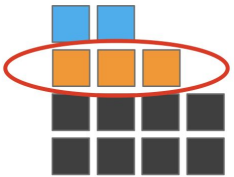
Parallel Execution Plan

- Quickly reinstantiate a new pipeline from a template when failures happen



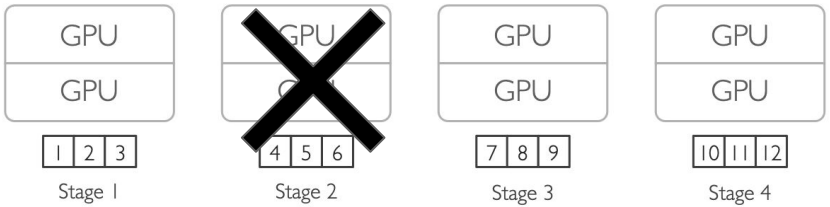
A pipeline instantiated from the template for 4 nodes



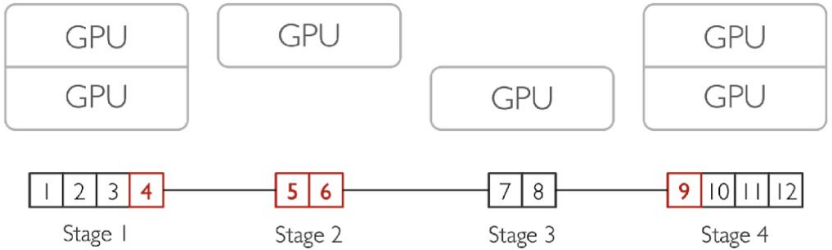


Parallel Execution Plan

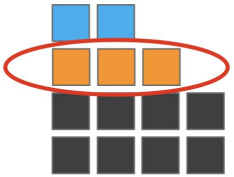
- Quickly reinstantiate a new pipeline from a template when failures happen



A pipeline instantiated from the template for 4 nodes

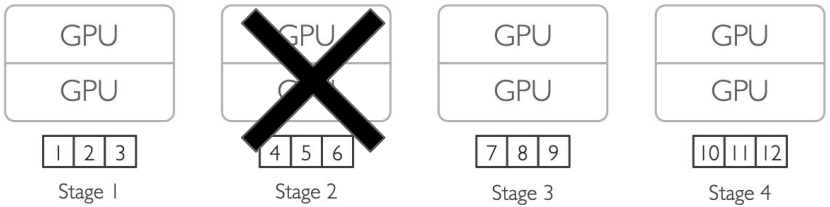


Instantiate a pipeline template for 3 nodes

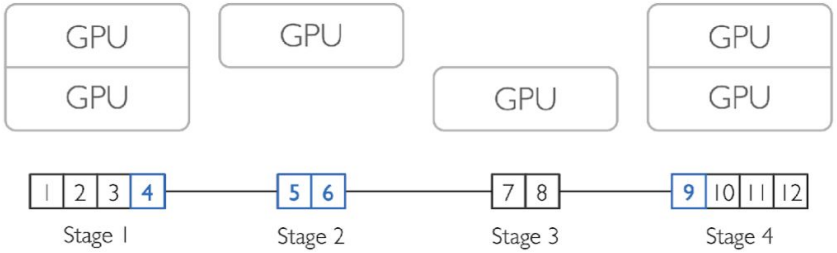


Parallel Execution Plan

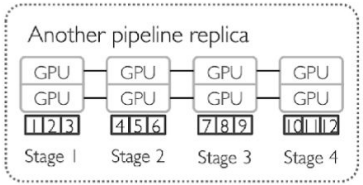
- Quickly reinstantiate a new pipeline from a template when failures happen

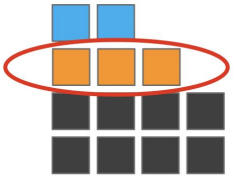


A pipeline instantiated from the template for 4 nodes



Copy missing layers from replica(s)





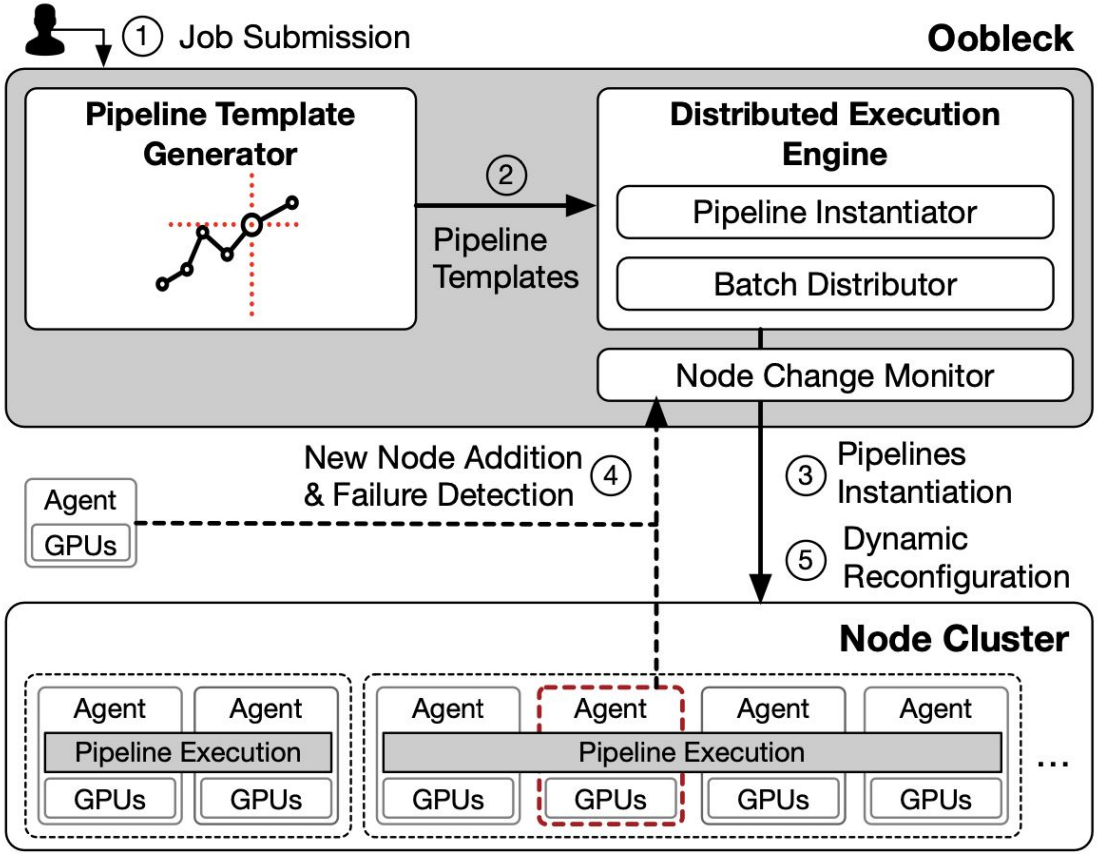


Figure 3. *Oobleck system overview.*

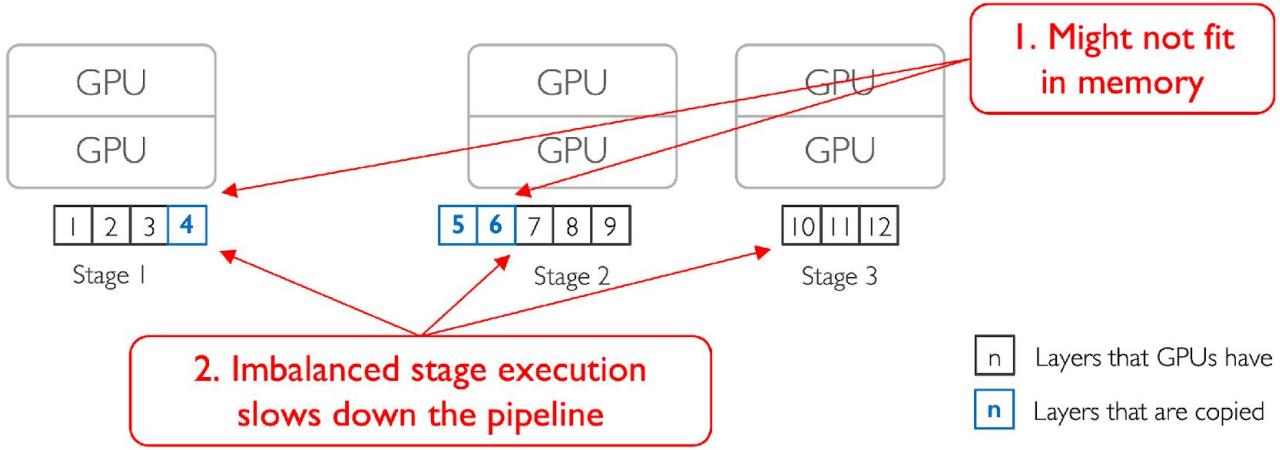
Generate pipeline templates

Instantiate pipelines from the pipeline templates

Pipeline reinstantiation when failures detected

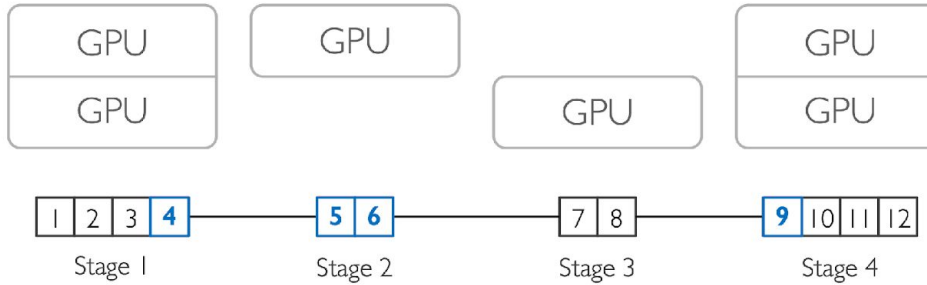
Re-instantiation vs Just Copying Layers

Copying lost layers to adjecent nodes without reinstantiation



VS

Pipeline reinstantiation



Challenges in Using Pipeline Templates

1. Determining # pipeline templates and # nodes for each template

   → Node Specification

Challenges in Using Pipeline Templates

1. Determining # pipeline templates and # nodes for each template

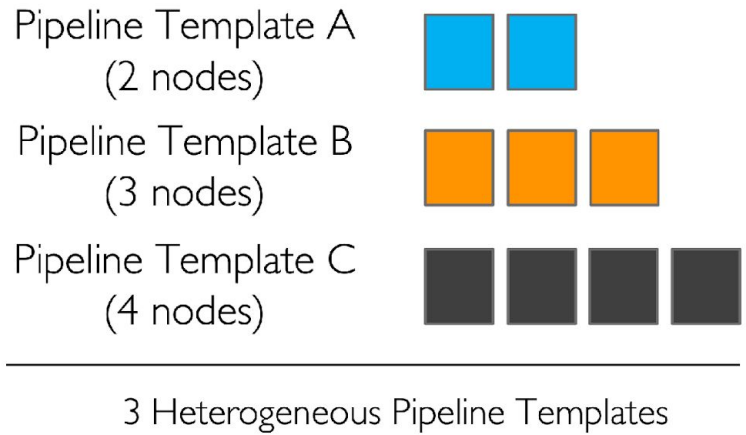
→ Node Specification

Train a model (required to have ≥ 2 nodes to train) with 13 nodes
How many pipeline templates do we need?

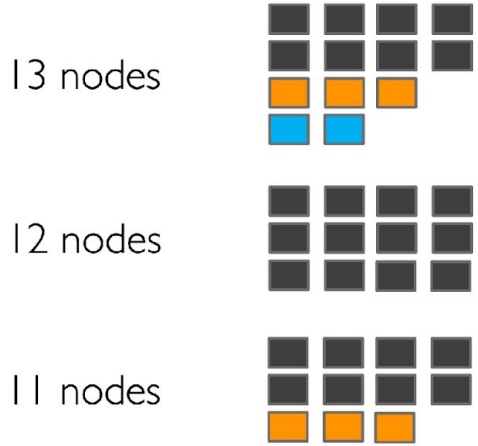
Pipeline Template for 2 Nodes	Pipeline Template for 3 Nodes	Pipeline Template for 4 Nodes	Pipeline Template for 5 Nodes
Pipeline Template for 6 Nodes	Pipeline Template for 7 Nodes	Pipeline Template for 8 Nodes	Pipeline Template for 9 Nodes
Pipeline Template for 10 Nodes	Pipeline Template for 11 Nodes	Pipeline Template for 12 Nodes	Pipeline Template for 13 Nodes

Node Specification

- Only need 3 templates (with nodes 2,3,4) to cover all execution plans with training nodes ***N*** between 2 and 13 inclusively.
- Formulated as Frobenius problem (sec 4.1.1)



3 Heterogeneous Pipeline Templates



Any $2 \leq N \leq 13$ can be represented with the set of pipeline templates

Challenges in Using Pipeline Templates

1. Determining # pipeline templates and # nodes for each template

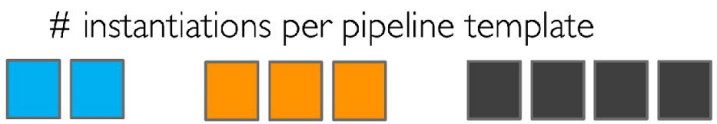
   → Node Specification

2. Determining number of pipelines to be instantiated from each template

   → Pipeline Instantiation

Pipeline Instantiation

- Use dynamic programming to enumerate all possible instantiation plans
- Estimate iteration time of every plans and pick the best one



	# instantiations per pipeline template			Total # nodes used
Plan 1	x1	x1	x2	13
Plan 2	x0	x3	x1	13
Plan 3	x5	x1	x0	13

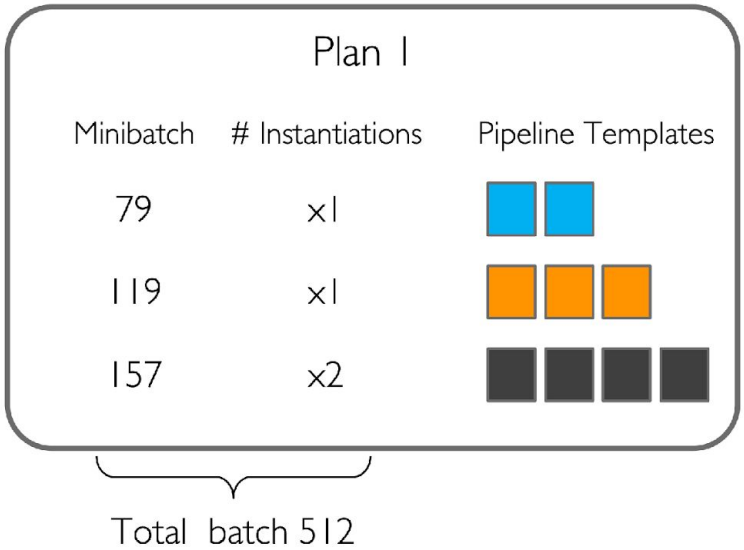
more options

Pipeline Instantiation: *Batch Distribution*

- Need to know batch size per pipeline to estimate iteration time
- Formulate finding batch distribution that minimizes overall iteration time as an integer optimization problem

Global batch
512

"Find batch size of each pipeline"



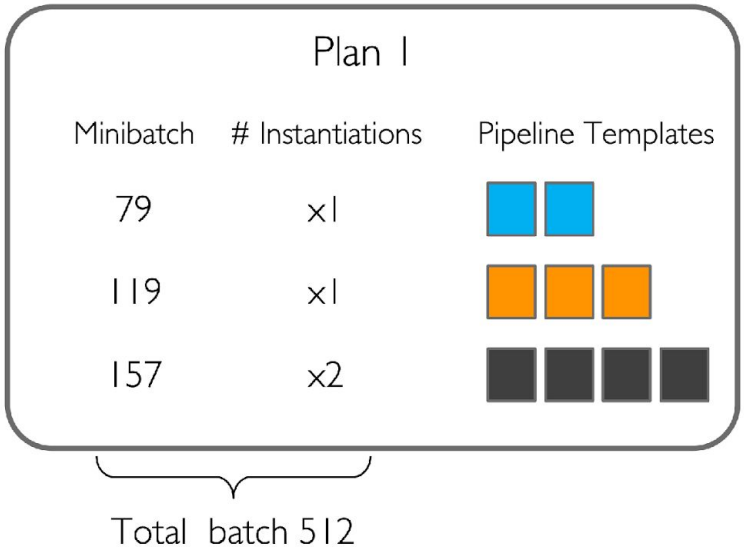
Pipeline Instantiation: *Batch Distribution*

- Need to know batch size per pipeline to estimate iteration time
- Formulate finding batch distribution that minimizes overall iteration time as an integer optimization problem

$$\text{minimize} \quad \sum_{i=0}^{x-1} (N_{b,i} T_i - \overline{N_b T})^2$$

$$\text{subject to} \quad \sum_{i=0}^{p-1} N_{b,i} b x_i = B,$$

$$N_{b,i} \in \mathbb{N}$$



Challenges in Using Pipeline Templates

1. Determining # pipeline templates and # nodes for each template

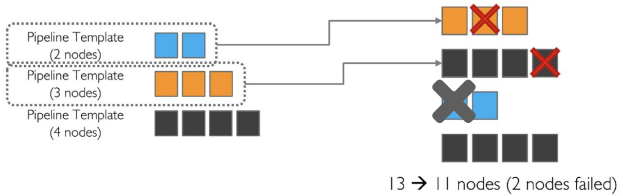
   → Node Specification

2. Determining number of pipelines to be instantiated from each template

   → Pipeline Instantiation

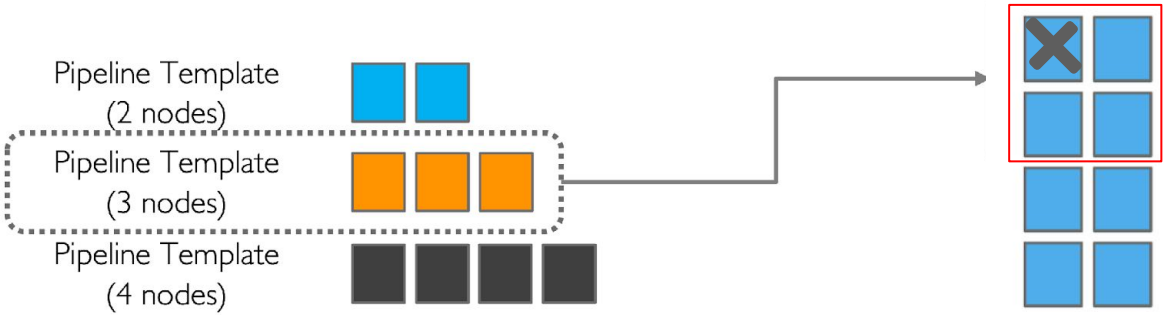
3. What if there is no feasible pipeline template to be instantiated?

   → Pipeline Merge



Pipeline Merge

- When no feasible pipeline template: **merge pipelines**
- **Provable guarantee** that Oobleck always has a template for merged pipeline



8 → 7 nodes (1 node failed)

Evaluation Setup

- Setup
  - Compare Bamboo, Varuna, and Oobleck
  - 30 NVIDIA A40 GPUs with 200Gbps Infiniband
  - Various size of models from BERT-Large (345M) to GPT-3 6.7b (6.7B)

- Questions
  - How much is Oobleck better than SOTAs (Bamboo and Varuna)?
  - Why Oobleck is better?

Simulated Environment

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

Real Spot Instance Environment

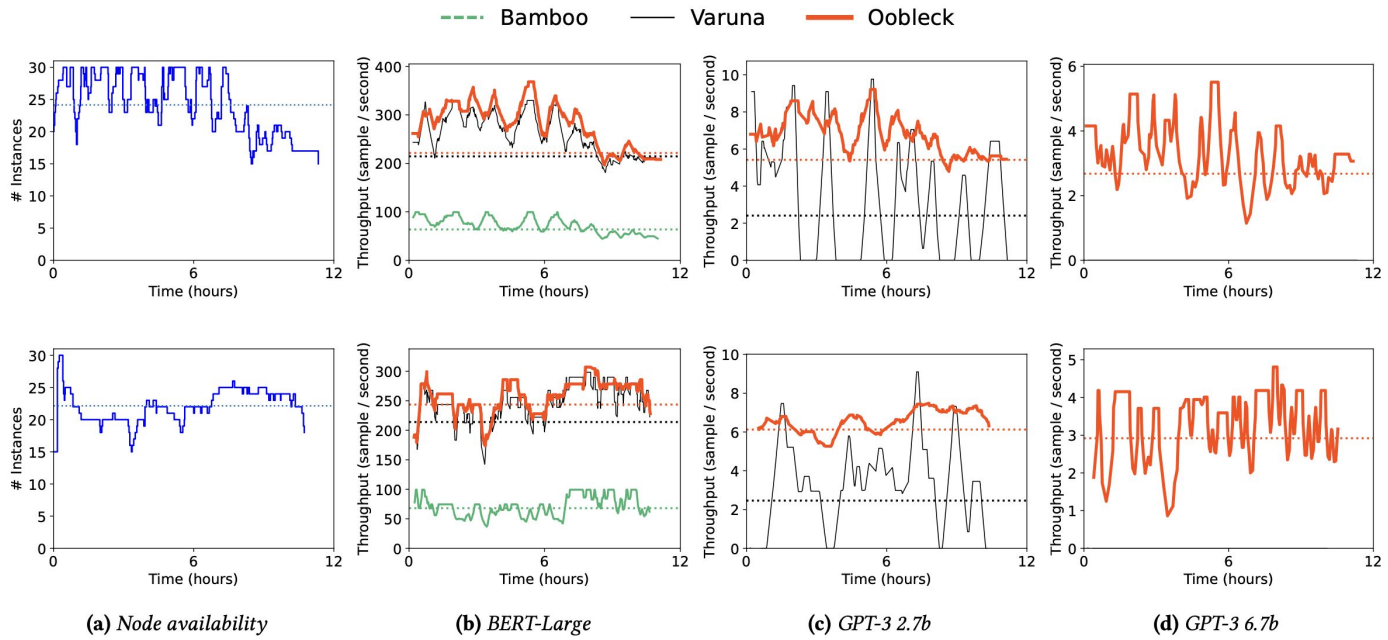
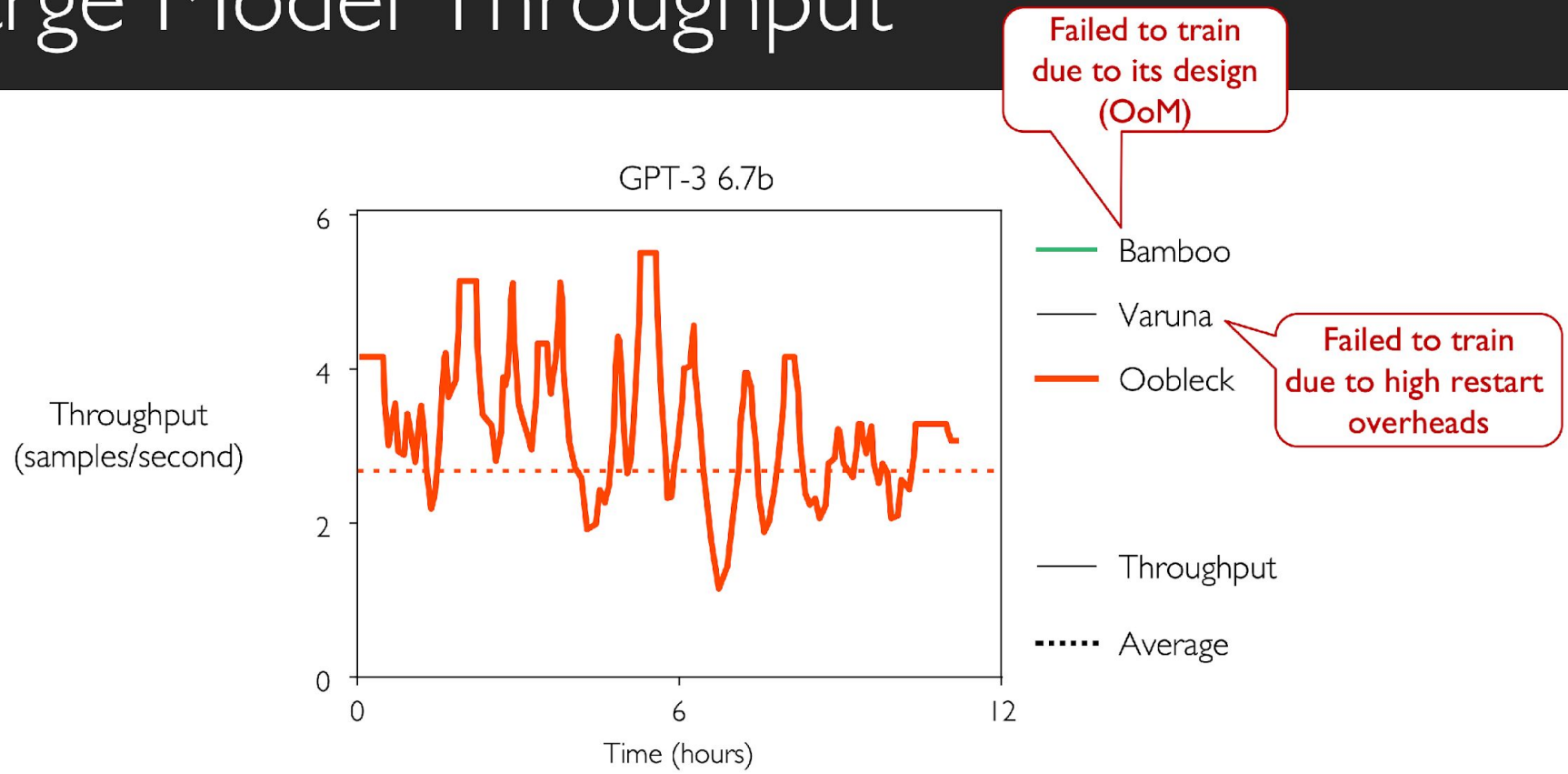


Figure 10. *Throughput changes in spot instances environment, EC2* `P3` *instances (top) and GCP* `a2-highgpu-1g` *instances (bottom), with various models. Note the different Y-axes scales for different models. Horizontal dotted lines represent average throughput. Bamboo could not run any GPT-3 models, while Varuna failed for GPT-3 6.7b.*

Large Model Throughput



Numbers in Bamboo EC2 p3 spot instance traces.

Thanks | QA

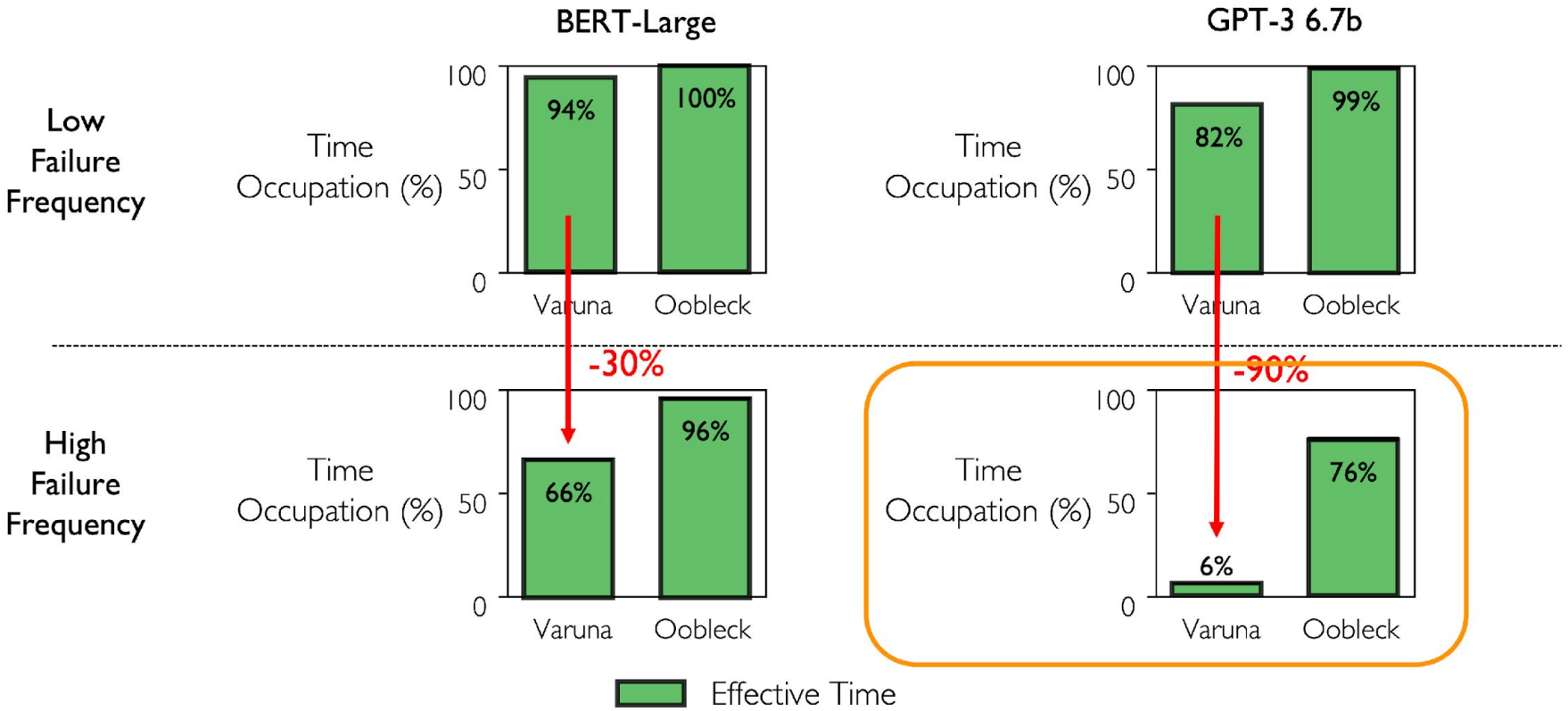
Existent Solutions

	Bamboo [1]	Varuna [2]	Oobleck (ours)
Guaranteed fault tolerance	✗ No guarantee for ≥ 2 simultanoues failures	? No formal fault tolerance guarantee	✓
High throughput	✗ High computational overheads	? Getting slower when recovery overheads are higher	✓
Fast recovery	✓ Dynamic reconfiguration without restart	✗ Full restart from the last checkpoint	✓

[1] John Thorpe et al. "Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large DNNs". NSDI'23
[2] Sanjith Athlur et al. "Varuna: Scalable, Low-cost Training of Massive Deep Learning Models". EuroSys'22

(From Insu's Slides)

Throughput vs Varuna



Numbers in failiure-controlled environment.

Time Breakdown

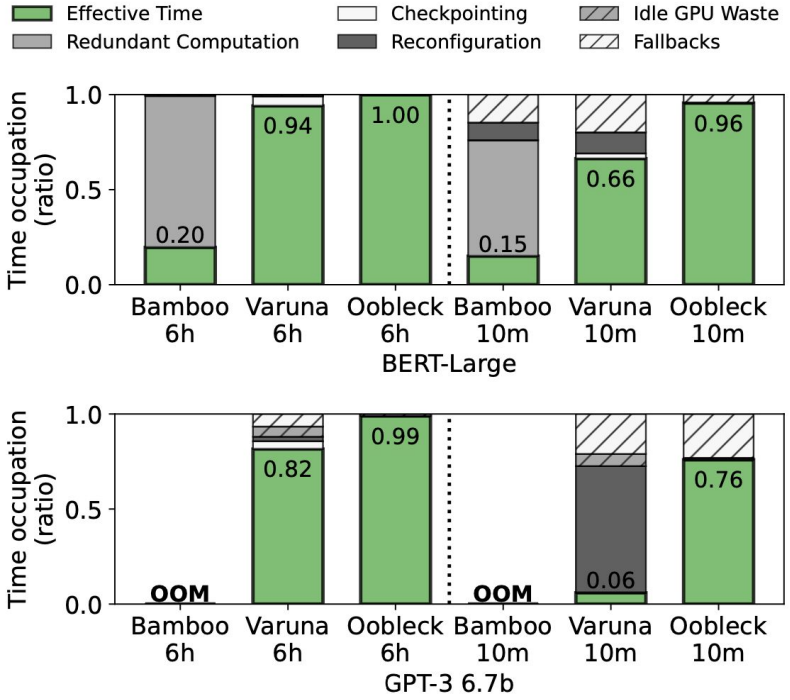


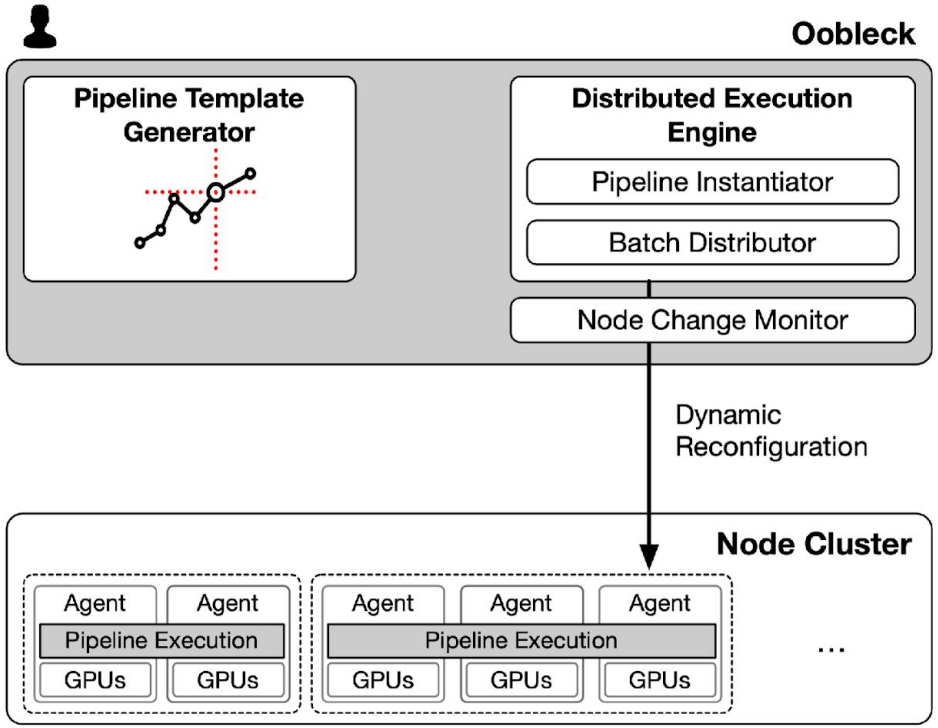
Figure 11. *Time occupation breakdown of Bamboo, Varuna, and Oobleck running BERT-Large and GPT-3 6.7b model.*

Pipeline Templates Computation Complex

Table 3. *Oobleck planning latency (in seconds) with various numbers of layers and nodes. BERT-Large, GPT-2, and GPT-3 Medium have 24 layers, while GPT-3 2.7b and 6.7b have 32 layers.*

# Nodes	# GPUs Per Node	# Layers			
		24	32	64	96
8	1	0.28	0.71	9.65	68.50
	4	0.41	1.15	11.58	74.56
	8	0.54	1.50	20.98	109.76
16	1	3.37	7.45	66.35	540.36
	4	4.56	10.41	108.10	649.67
	8	4.90	11.78	176.04	1,213.63
24	1	11.35	30.11	262.47	1,477.54
	4	14.78	45.80	472.53	2,153.84
	8	15.59	49.25	520.08	3,297.92

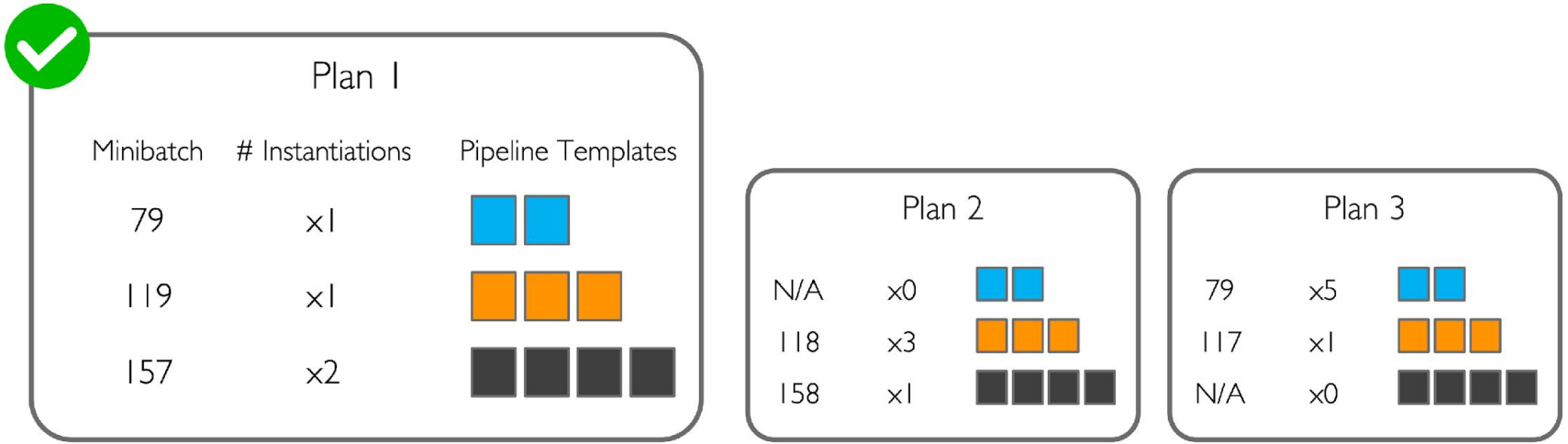
Oobleck Architecture Workflow



1. Generate pipeline templates
2. Instantiate pipelines from the pipeline templates
3. Pipeline reinstantiation when failures detected

Batch Distribution

- Estimate iteration time of every plans and pick the best one



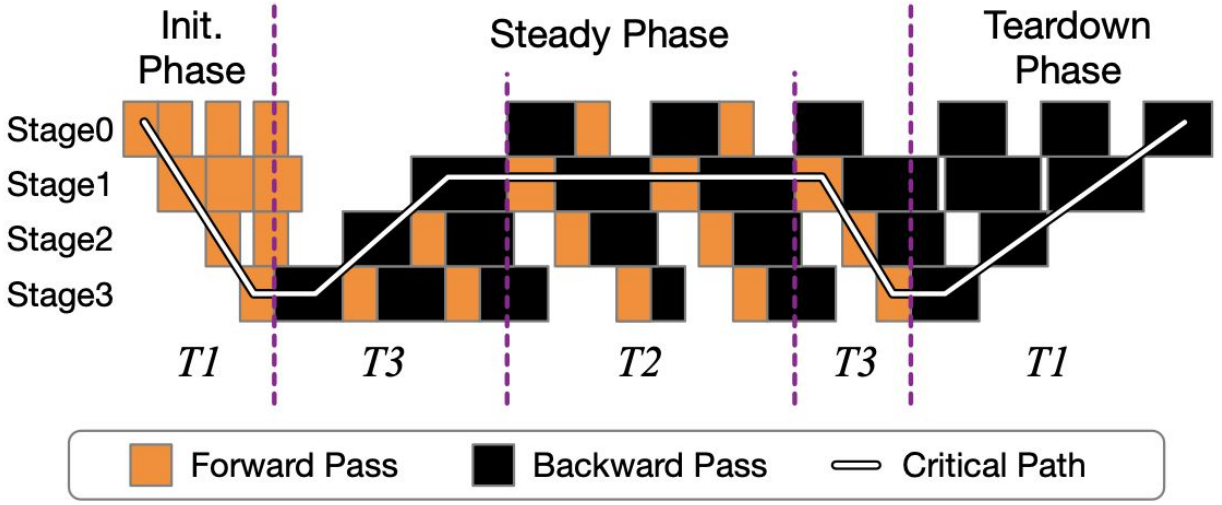
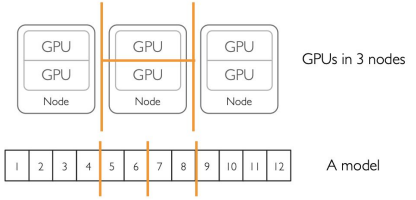


Figure 5. *1F1B pipeline execution breakdown (T1, T2, T3)*



References

- Oobleck Presentation
- CMU 15-418 Lecture 25: Parallel Deep Learning (Model & Pipeline Parallelism)
- GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism