



Parcae: Proactive, Liveput-Optimized DNN Training on Preemptible Instances

NSDI'24

Jiangfei Duan*, Ziang Song*, Xupeng Miao*, Xiaoli Xi,
Dahua Lin, Harry Xu, Minjia Zhang, Zhihao Jia

**Carnegie
Mellon
University**



香港中文大學
The Chinese University of Hong Kong

 **ByteDance**

UCLA

 **Microsoft**

Presenter: Yujie Chen

2025.4.18

Author

研究方向

- **ML编译与部署**

Machine Learning Compilation for LLMs: 通用LLM部署方案。

- **并行化策略与超参数优化**

FlexFlow: 自动发现DNN训练的高效并行化策略。

Hyperband & ASHA: 基于原理化早停方法的超参数优化框架。

- **可扩展学习系统**

XGBoost: 可扩展的分布式树提升系统, 广泛应用于工业与科研。

- **LLM系统**

XGrammar: 高效、灵活且可移植的LLM结构化生成框架。

TidalDecode: 面向LLM解码的稀疏Attention框架。

FlexFlow Serve: 面向LLM的低延迟高性能在线服务系统。

Carnegie
Mellon
University



Tianqi Chen
Assistant Professor

<https://catalyst.cs.cmu.edu/>

Contents

- **Background**
 - **Review**
 - **Design**
 - **Evaluation**
 - **Thinking**
-

Background

LLM Training Is Expensive



\$3.9 Million



\$2-5 Million



\$4.6 Million

Background

Spot Instances Reduce Training Cost



up to 10× Savings



up to 9× Savings



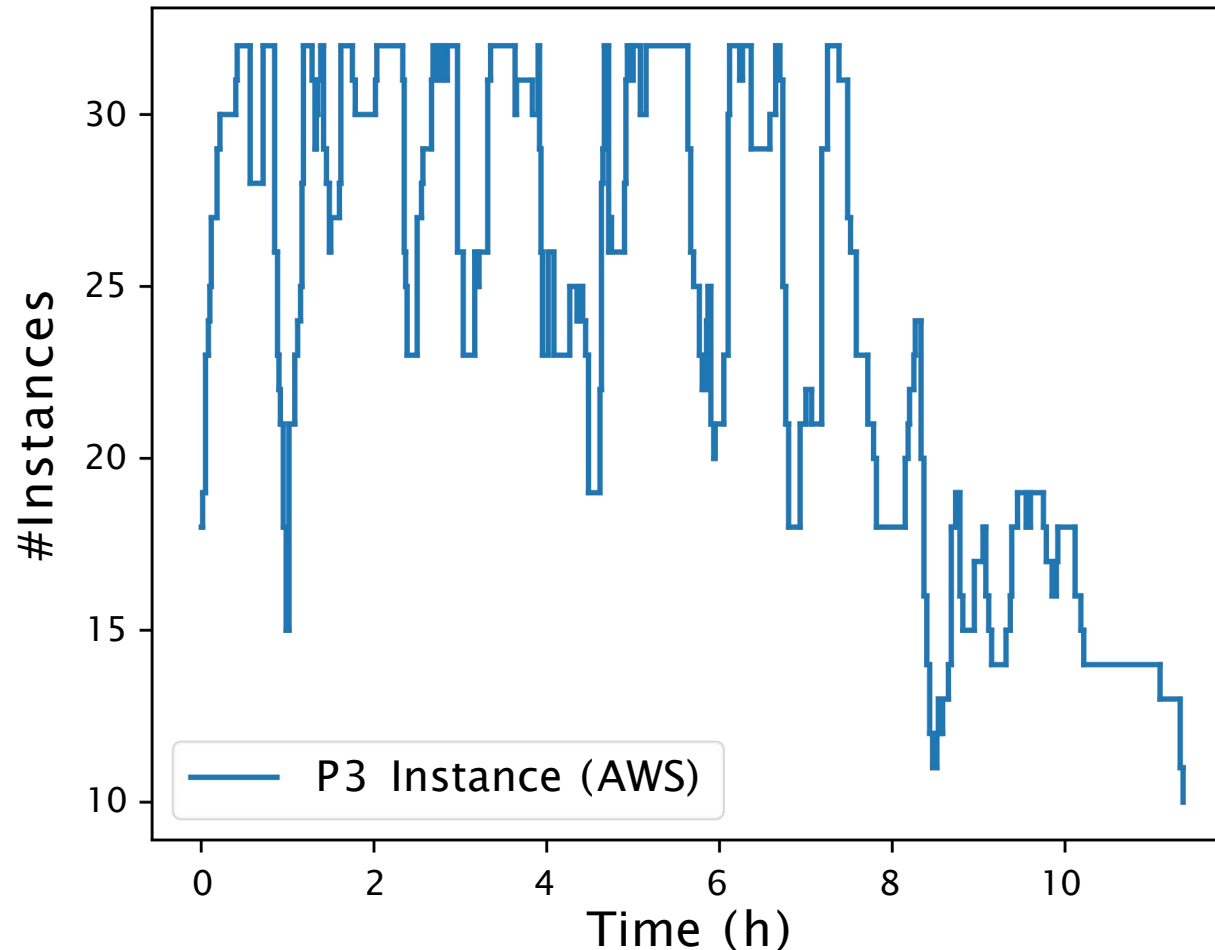
up to 3× Savings



2× Savings

Background

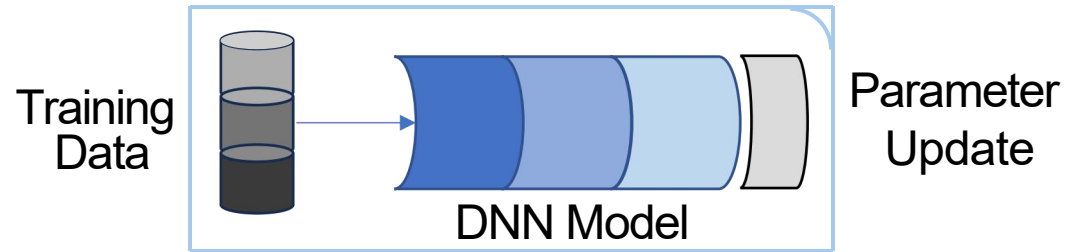
Spot Instances Reduce Training Cost,
But Are Preemptible



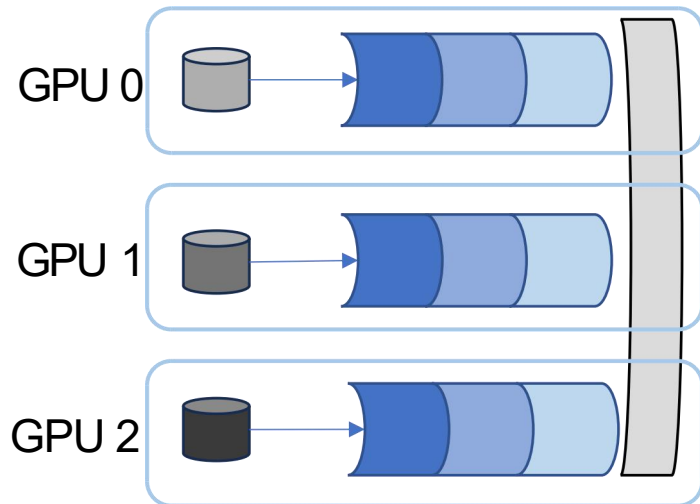
Preemption once every
5 minutes,
worst case **1** minute

Background

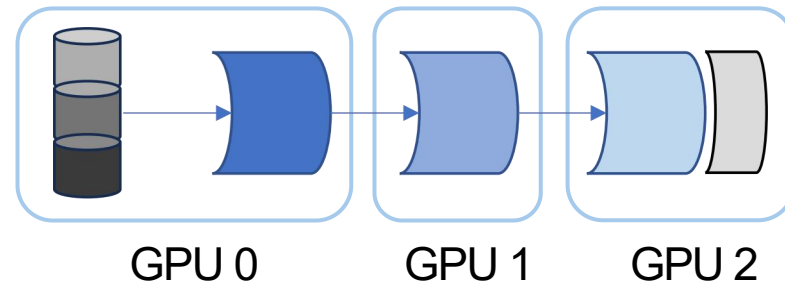
Distributed DNN Training



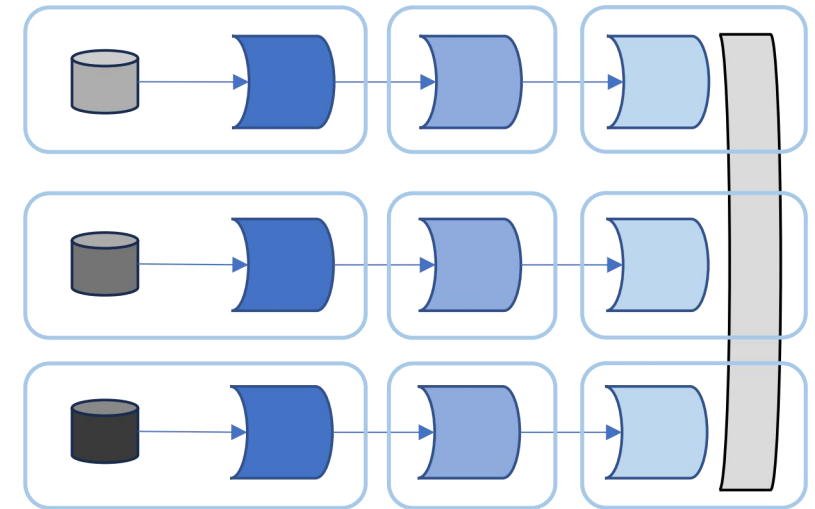
Single GPU Training



Data Parallelism



Pipeline Parallelism



Hybrid Parallelism

Background

- Checkpoint-Based Systems

Uses checkpoints to maintain model states during training.

Example: Varuna periodically saves model states to persistent storage.

Varuna: scalable, low-cost training of massive deep learning models.(EuroSys22)

- Redundancy-Based Systems

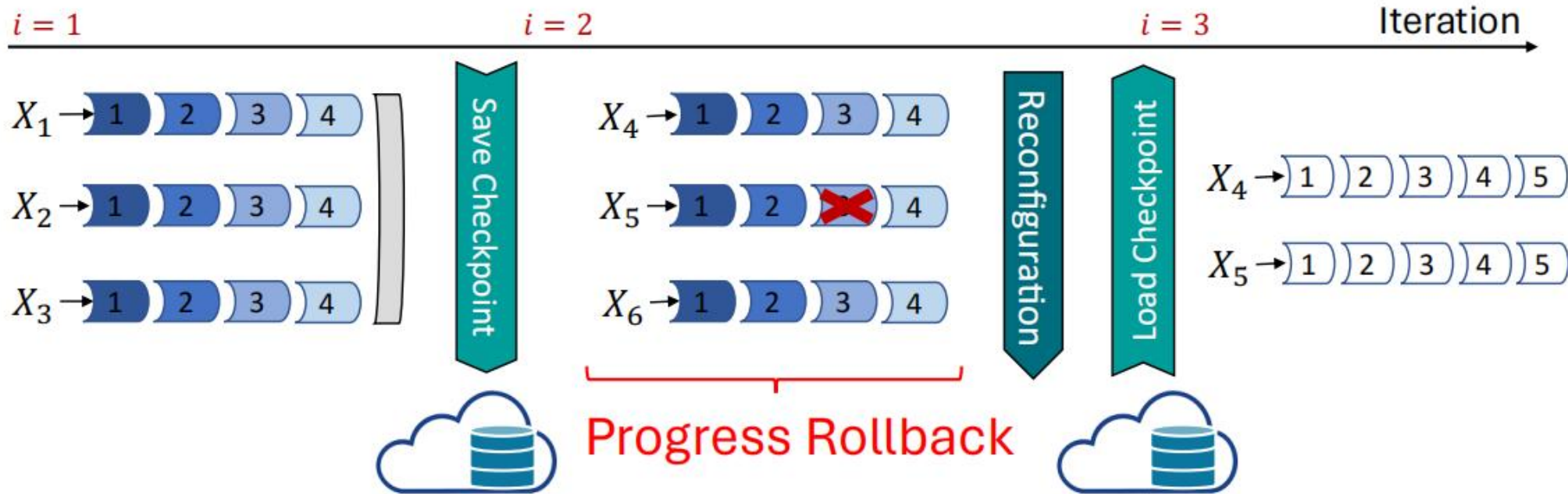
Uses redundant computation for resilience.

Example: Bamboo replicates DNN computations across instances.

Bamboo: Making preemptible instances resilient for affordable training of large DNNs.(NSDI23)

Background

•Checkpoint-Based: Varuna (EuroSys '22)



✓Pros:

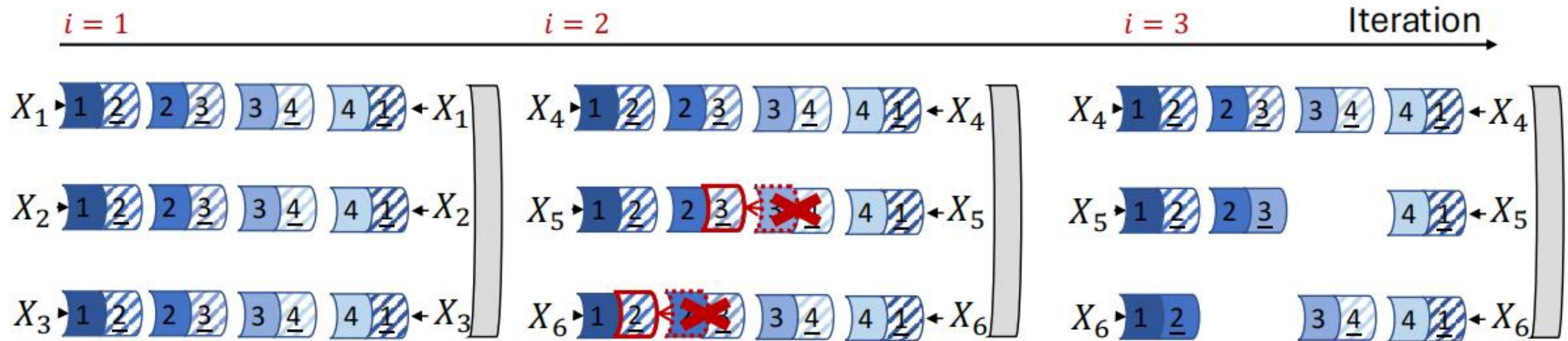
- Simple implementation
- efficient when preemptions are rare

✗Cons:

- High IO overhead
- Frequent rollbacks under high preemption slow progress

Background

- Redundancy-based: Bamboo (NSDI '23)



Redundant Computation and Memory Overhead

✓ Pros:

- Higher throughput under high preemption
- No frequent state saving

✗ Cons:

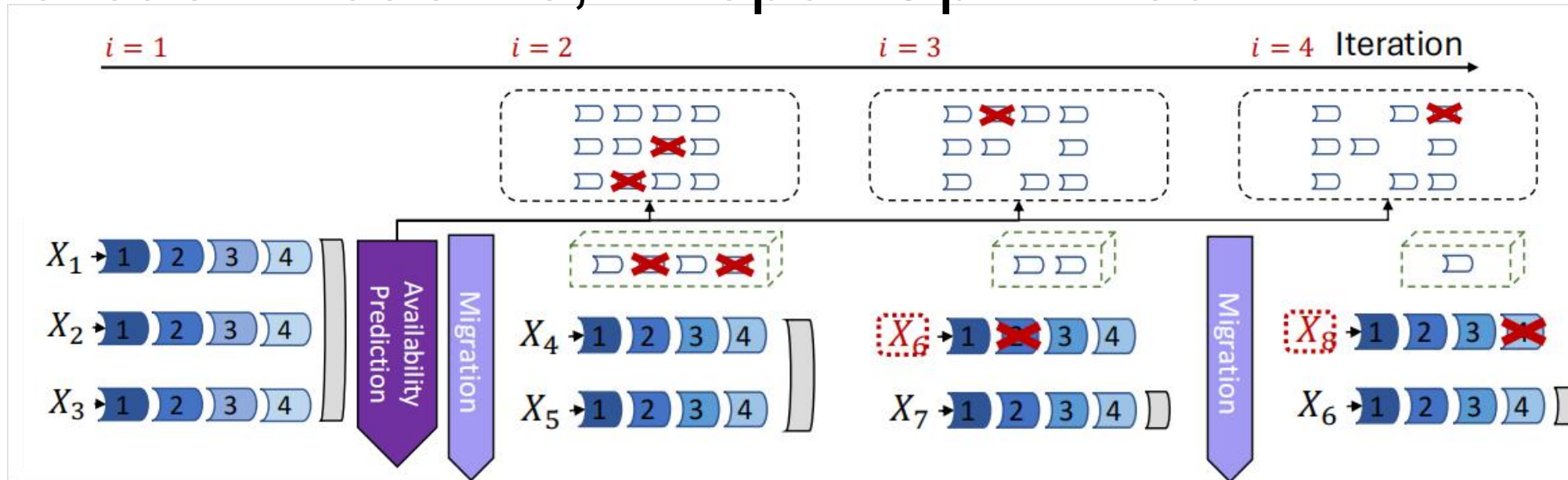
- Redundant computing reduces efficiency
- Increased GPU memory usage

Parcae: Proactive, Liveput-Optimized

- **Proactive**
 - Parcae employs a proactive strategy to manage DNN training on spot instances.
 - Anticipates and mitigates preemption impacts before they occur.
- **Liveput**
 - Liveput measures expected training throughput under various preemption scenarios.
 - Guides optimal training strategies to maximize efficiency in preemptible environments.

Review

Parcae: Proactive, Liveput-Optimized



Proactively plan for future preemptions to maximize
preemption-aware throughput (Liveput)

➤ Predicting

➤ Maximize liveput

➤ Handle Preemption

Design

Statistical Availability Predictor

- Relies on past preemption/allocation records to forecast future availability.

Problem Formulation:

- Symbols:** Timeline split into intervals (N_i, N_i^+, N_i^-)

- N_i : Number of available instances

- N_i^+ : Newly allocated instances

- N_i^- : Preempted instances

$$N_i = N_{i-1} + N_i^+ - N_i^- \quad (i > 0)$$

$$(N_i, \dots, N_{i+I-1}) = \text{PREDICTION}(N_{i-H}, \dots, N_{i-1})$$

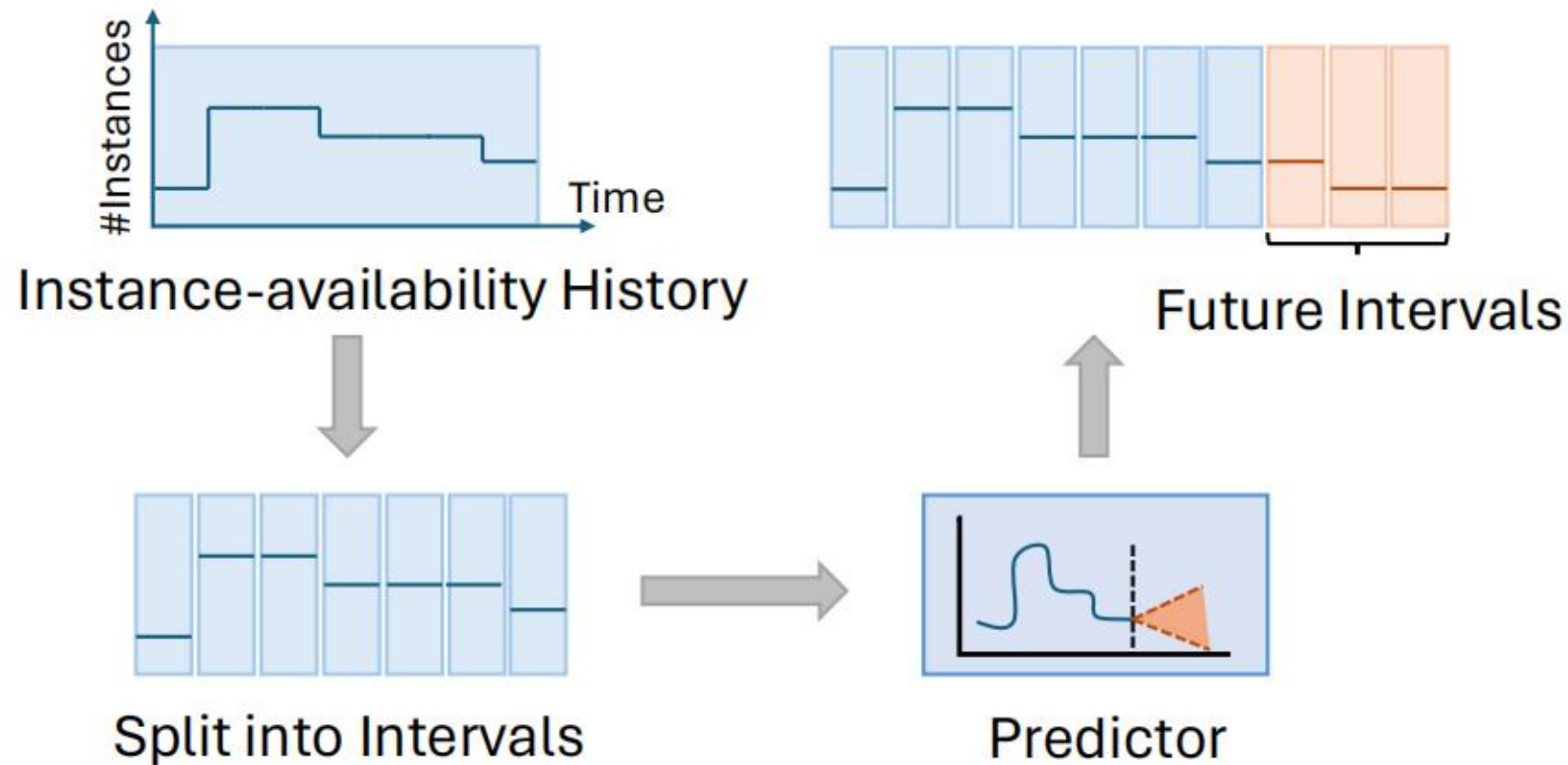
$$N_i^+ = \max(0, N_i - N_{i-1})$$

$$N_i^- = \max(0, N_{i-1} - N_i)$$

Cloud does not preempt existing instances and allocate new instances at the same time.

Design

Statistical Availability Predictor

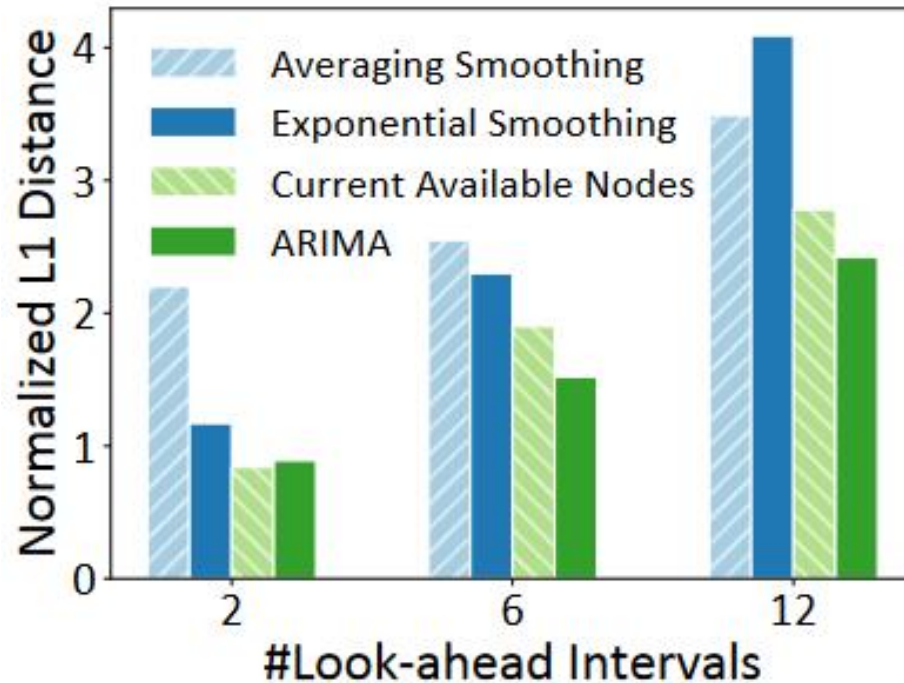


Time-series forecasting problem!

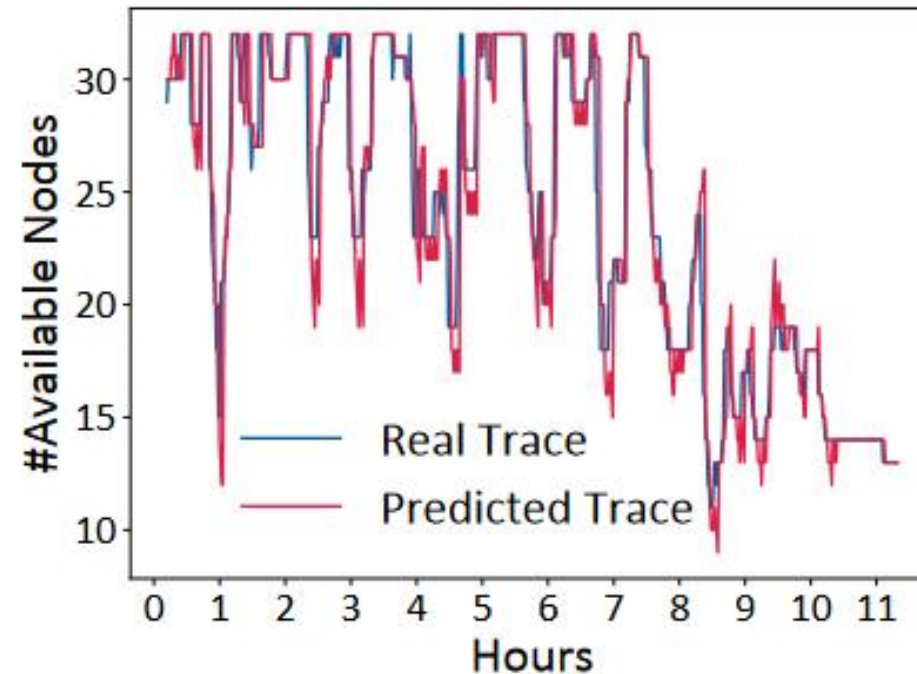
Design

Statistical Availability Predictor

- Limited input data prevents Parcae from using complex prediction models



(a) ARIMA vs. other models

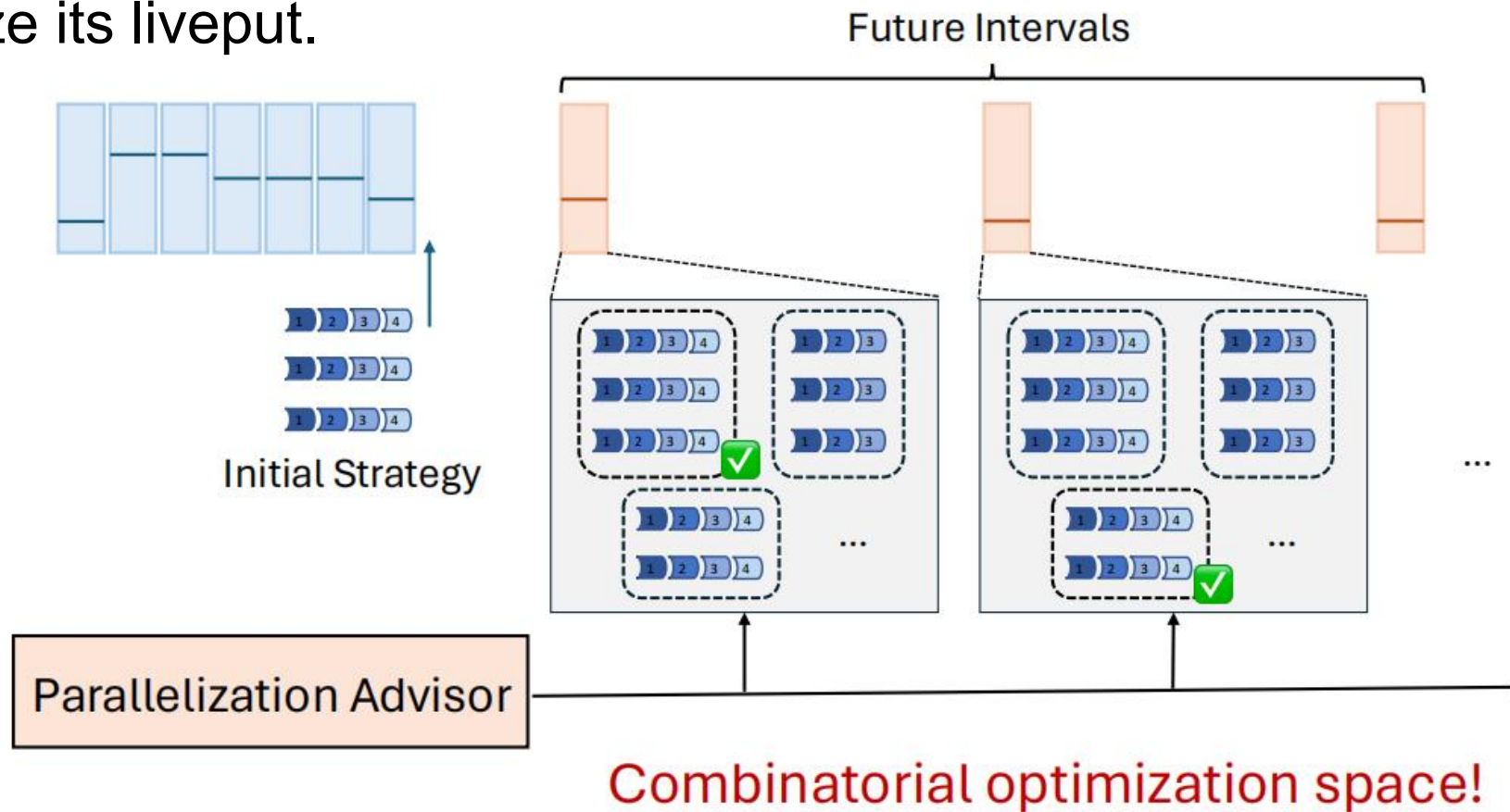


(b) ARIMA-predicted vs. real trace

Design

Liveput Optimizer

- determines the parallel configurations of training DNN model on spot instances to maximize its liveput.



Design

Liveput:

Liveput measures expected training throughput under various preemption scenarios.

$$\text{LIVEPUT}(D, P, \mathcal{V}) = \mathbb{E}_{\underbrace{\vec{v} \sim \mathcal{V}}_{\text{Preemption Scenario}}} [\underbrace{\text{THROUGHPUT}(D_{\vec{v}}, P_{\vec{v}})}_{\text{Parallelization Strategy after Preemption}}]$$

D: number of data-parallel pipelines.

P: number of pipeline stages

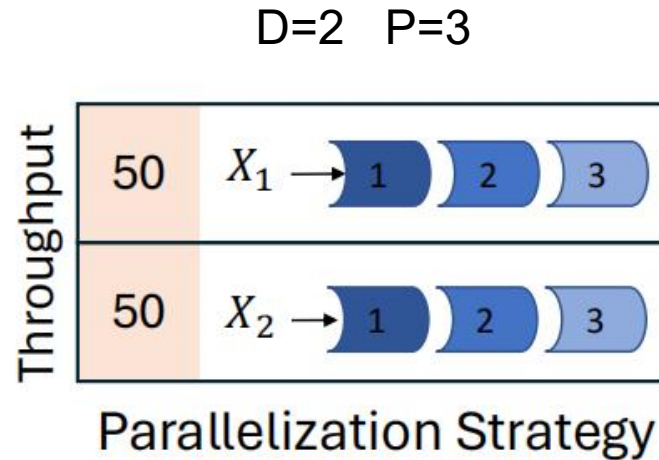
V: $\{0, 1\}^{D \times P} \rightarrow [0, 1]$, the probability distribution of all preemption scenarios

\vec{v} : an preemption indicator vector, $\vec{v}_k = 1$ if instance k will be preempted.

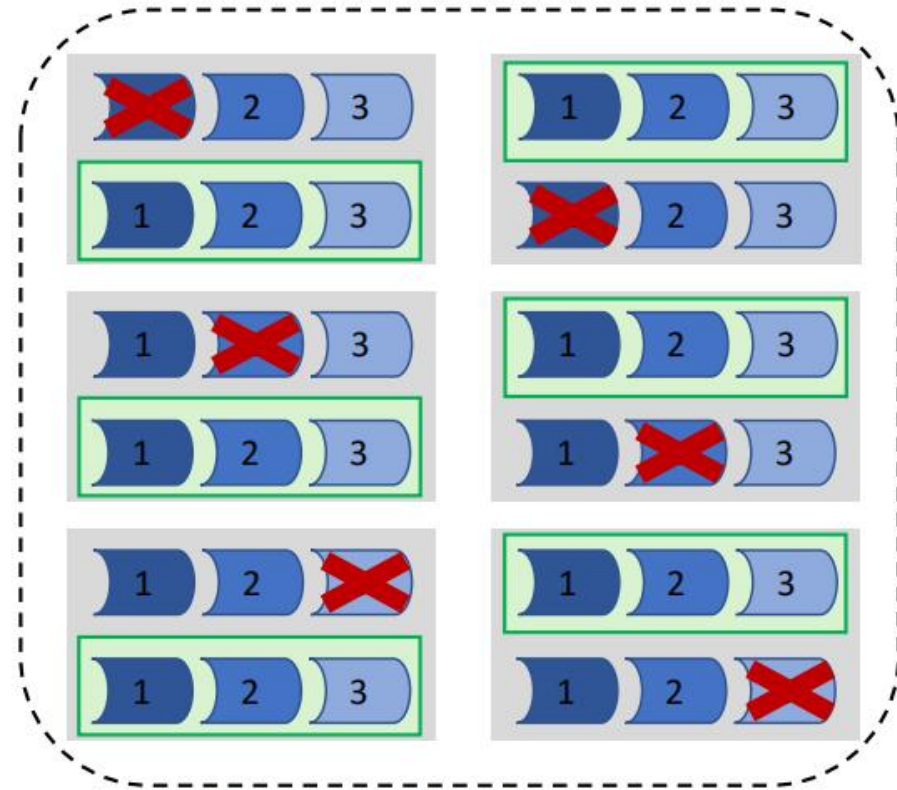
Design

Liveput Example

$$\text{LIVEPUT}(D, P, \mathcal{V}) = \mathbb{E}_{\vec{v} \sim \mathcal{V}}[\text{THROUGHPUT}(D_{\vec{v}}, P_{\vec{v}})]$$













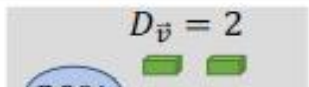




1 preemption








$$\text{LIVEPUT (1 preemption)} = 100\% \times 50 = 50$$










Design

Configurations	#Preemptions	Preemption Scenarios Distribution	THROUGHPUT	LIVEPUT
$D = 2 \quad P = 3$	0	100%  $D_{\bar{v}} = 2$	100 	$100\% \times 2 \times 50 = 100$
50 $X_1 \rightarrow$ 	1	100%  $D_{\bar{v}} = 1$		$100\% \times 1 \times 50 = 50$
50 $X_2 \rightarrow$ 	2	<div> $D_{\bar{v}} = 1$ 40%  </div> <div> $D_{\bar{v}} = 0$ 60%  </div>		$40\% \times 1 \times 50 = 20$
$D = 3 \quad P = 2$	0	100%  $D_{\bar{v}} = 3$	90 	$100\% \times 3 \times 30 = 90$
30 $X_1 \rightarrow$ 	1	100%  $D_{\bar{v}} = 2$		$100\% \times 2 \times 30 = 60$
30 $X_2 \rightarrow$ 	2	<div> $D_{\bar{v}} = 2$ 20%  </div> <div> $D_{\bar{v}} = 1$ 80%  </div>		$20\% \times 2 \times 30$ $+ 80\% \times 1 \times 30 = 36$ 

Design

Configurations	#Preemptions	Preemption Scenarios Distribution	THROUGHPUT	LIVEPUT
$D = 2 \quad P = 3$	0	 $D_{\vec{v}} = 2$	100	$100\% \times 2 \times 50 = 100$ ✓
50 $X_1 \rightarrow$   	1	 $D_{\vec{v}} = 1$		$100\% \times 1 \times 50 = 50$

Maximizing Throughput Is Sub-Optimal

$D = 3 \quad P = 2$	0	 $D_{\vec{v}} = 3$	90	$100\% \times 3 \times 30 = 90$
30 $X_1 \rightarrow$  	1	 $D_{\vec{v}} = 2$		$100\% \times 2 \times 30 = 60$ ✓
30 $X_2 \rightarrow$  		 $D_{\vec{v}} = 2$ and $D_{\vec{v}} = 1$		$20\% \times 2 \times 30 + 80\% \times 1 \times 30 = 36$ ✓
30 $X_3 \rightarrow$  	2			

Design

Liveput Optimizer

- **Objective:** discover a sequence of parallel configurations to **maximize the liveput**

$$\Phi(\mathbf{D}, \mathbf{P} \mid \mathbf{N}) = \sum_{i=0}^{I-1} \phi(D_i, P_i, N_i \mid D_{i+1}, P_{i+1}, N_{i+1}) \implies \arg \max_{\mathbf{D}, \mathbf{P}} \Phi(\mathbf{D}, \mathbf{P} \mid \mathbf{N})$$

- Single time period Liveput:

$$\phi(D_i, P_i, N_i \mid D_{i+1}, P_{i+1}, N_{i+1}) = \mathbb{E}_{\vec{v}_{i+1}} [\text{LIVEPUT}(D_{i+1}, P_{i+1} \mid \vec{v}_{i+1}) \times T_{\text{eff}}]$$

- T_{eff} : effective training time after migrations; T_{mig} : migration overhead

$$T_{\text{eff}} = T - T_{\text{mig}}(D_i, P_i, D_{i+1}, P_{i+1} \mid \vec{v}_{i+1}),$$

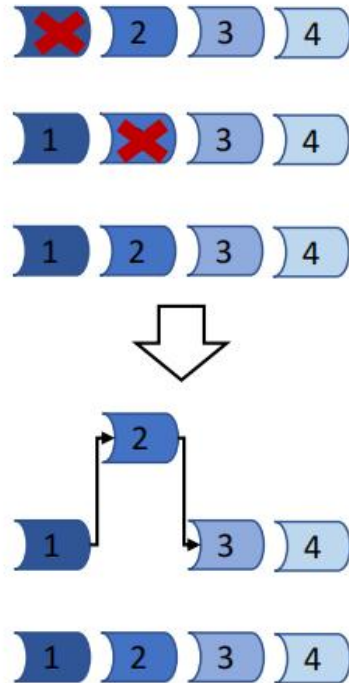
$$\text{LIVEPUT}(D, P, \mathcal{V}) = \mathbb{E}_{\vec{v} \sim \mathcal{V}} [\text{THROUGHPUT}(D_{\vec{v}}, P_{\vec{v}})]$$

Dynamic Programming:

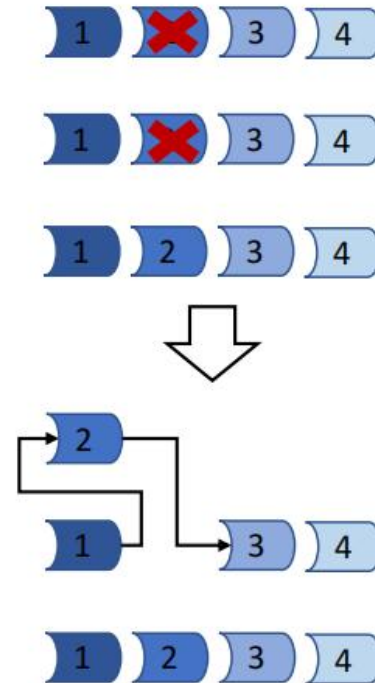
$$F(i+1, D_{i+1}, P_{i+1}) = \max_{\forall D_i \times P_i \leq N_i} \left\{ \begin{array}{l} F(i, D_i, P_i) + \\ \phi(D_i, P_i, N_i \mid D_{i+1}, P_{i+1}, N_{i+1}) \end{array} \right\} \quad (D_i \times P_i \leq N_i)$$

Design

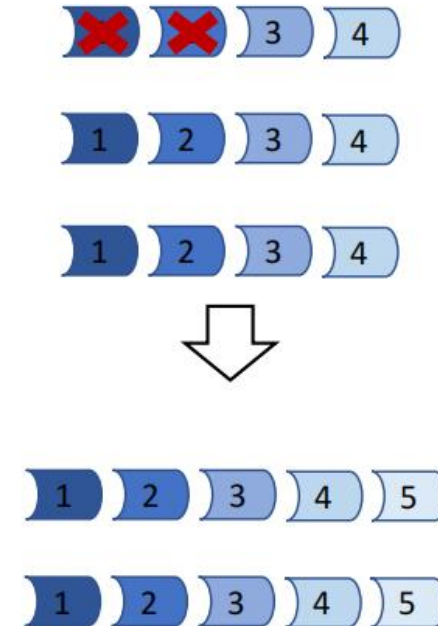
Handle Preemption



Intra-stage Migration



Inter-stage Migration



Pipeline Migration

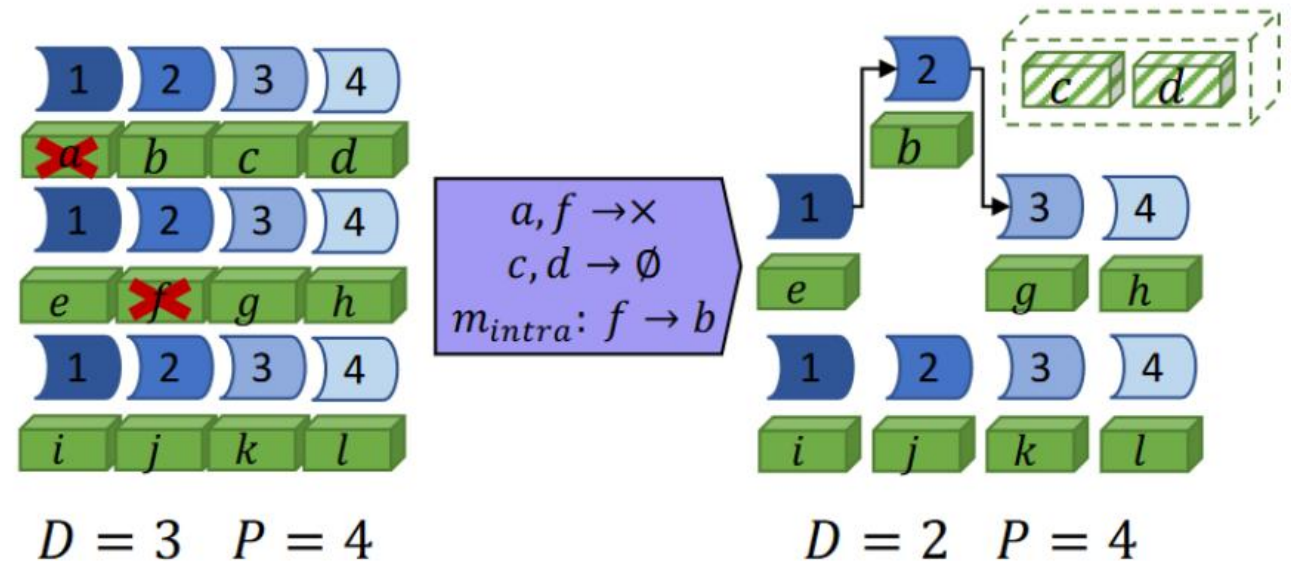
The actual migration decisions are finalized when preemptions really happen.

Design

Handle Preemption - Intra-stage Migration

- Principle: Instances within the same stage share parameters and can seamlessly substitute.
- Characteristics:
 - No parameter transfer needed
 - Only update routing
 - Highly efficient recovery

(a) Intra-stage Migration

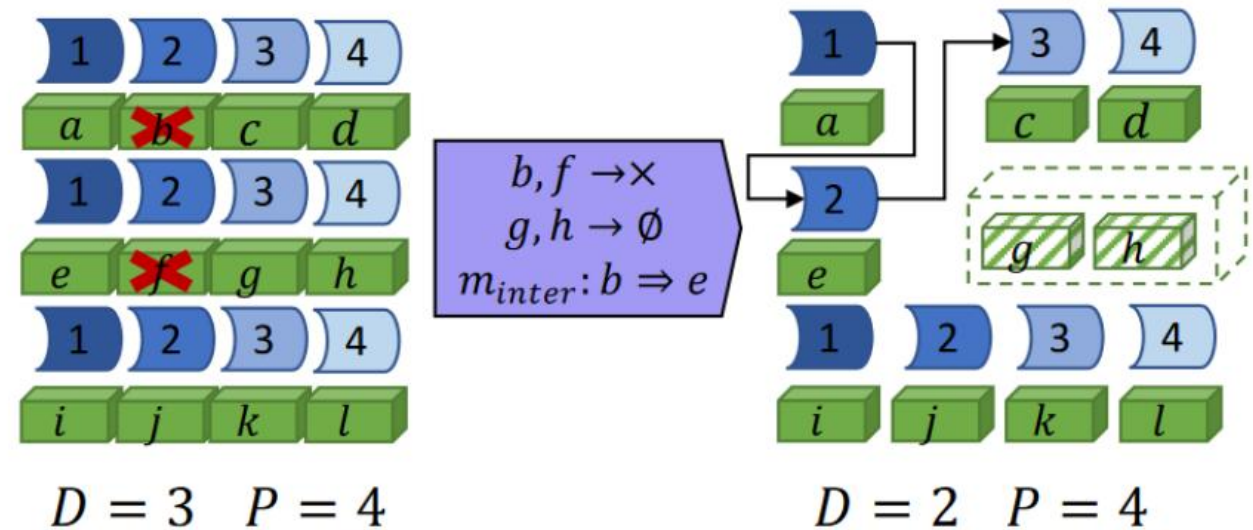


Design

Handle Preemption - Inter-stage Migration

- Principle: When no idle instance exists in a stage, migrate across stages and update parameter.
- Characteristics:
 - Requires parameter transfer
 - Incurs migration cost
 - Offers higher resilience

(b) Inter-stage Migration

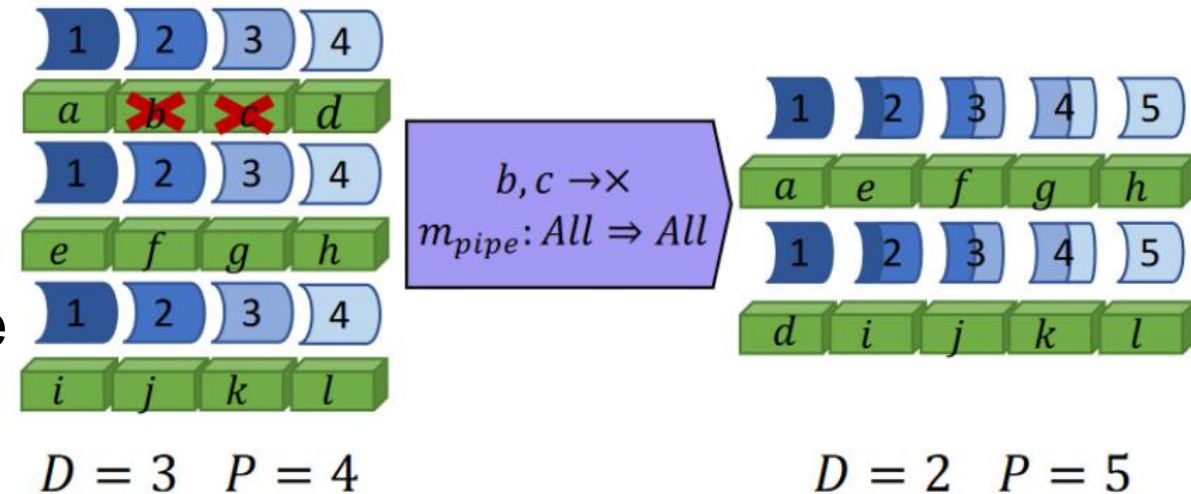


Design

Handle Preemption - Pipeline Migration

- Principle: Under severe preemptions, repartition model (e.g., 4→5 stages)
- Characteristics:
 - Highest overhead
 - Involves re-partitioning
 - parameter broadcast
 - Suited for large-scale resource change

(c) Pipeline Migration



Design

Liveput Optimizer : migration overhead

- Considers instance network topology for each preemption scenario.
- Uses an **α - β model** to estimate communication costs accurately.

α - β Model (Communication Cost)

α : Fixed startup latency

β : Per-byte transfer cost

The model estimates the time required to send a message of size **m** bytes as follows:

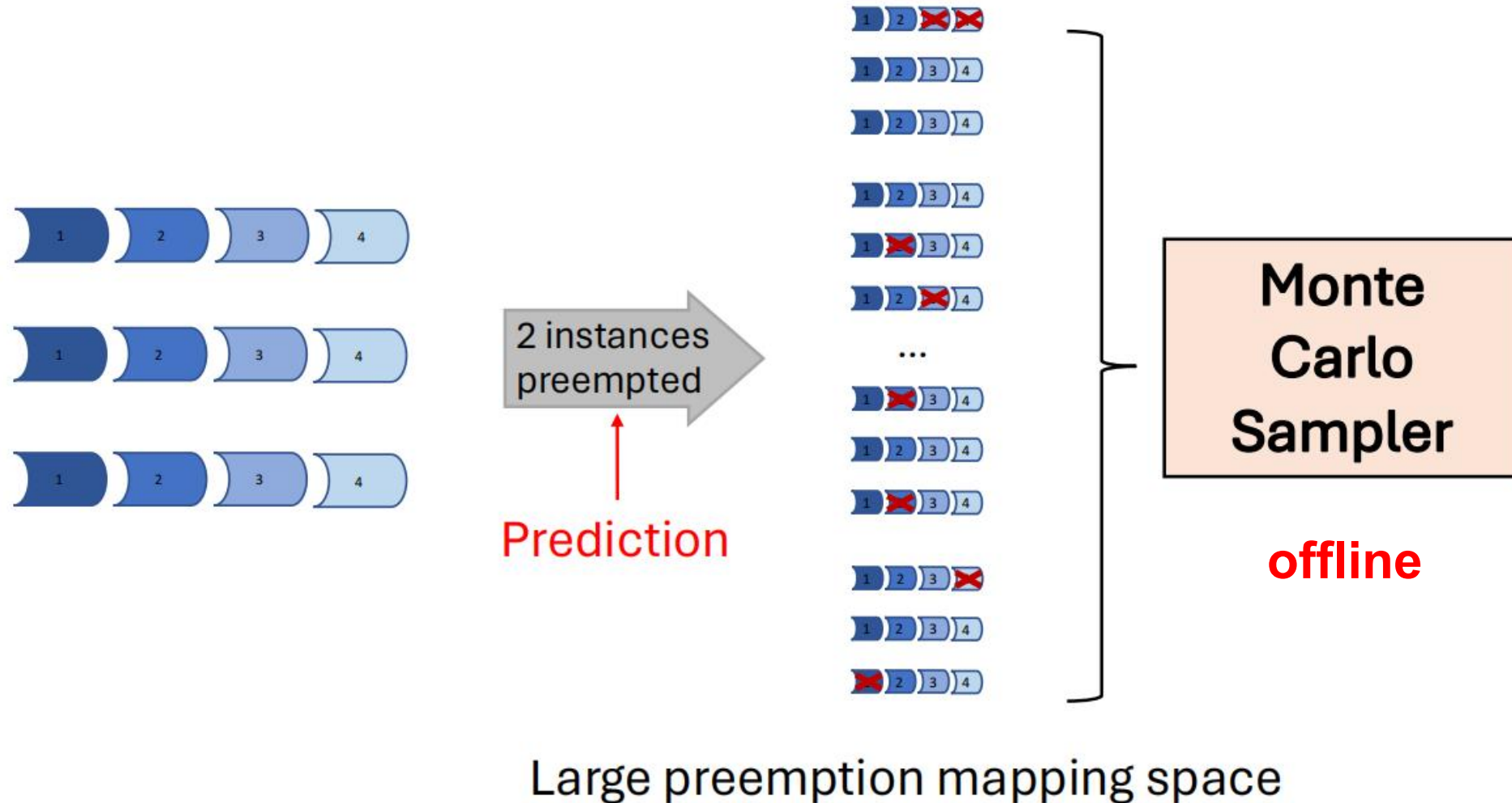
$$T = \alpha + \beta \times m$$

Table 4: Migration costs in our experiments on AWS.

Cost Terms	Magnitude (s)	Interfering Factor
Start process	< 1	Instance state
Rendezvous	0 ~ 10	
Init CUDA context	0 ~ 10	
Load data	0 ~ 10	Dataset
Build model	0 ~ 10	Model, Configuration
Update comm. groups	0 ~ 20	
Model states transfer	0 ~ 60	Model, Configuration Preemption Scenario

Design

Preemption Mapping Sampling



Exception Handling-Parallelization adaptation

Dynamically align parallel configuration with actual spot-instance availability.

- Predictive optimization may mismatch with real-time preemptions.
 - If actual instances $>$ predicted N_i : \rightarrow Add pipelines
 - If actual instances $<$ predicted N_i : \rightarrow Drop pipelines
 - If not enough instances for a full pipeline \rightarrow Repartition model into fewer stages
- Preserves pipeline depth when feasible
- Ensures configuration remains deployable with minimal overhead

Exception Handling-Fault tolerance

Exist rare cases where the migration strategies do not work.

- Scenarios
 - All instances of a specific stage are preempted
 - Available instances < minimum pipeline depth P
- Solution:
 - In-Memory Lightweight Checkpointing
 - Maintains model state(e.g.,parameters and optimizer states) in DRAM on **cheap on-demand CPU instances**.
 - Uses **gradient synchronization** instead of full state transfer, reducing communication by $\sim 5\times$.
 - If below feasible resource threshold \rightarrow Wait for spot instances to repopulate

Design

▷ ParcaeScheduler

```

1: function MIGRATIONMANAGER( $D_0, P_0$ )
2:   for  $i$  in 1, 2, 3, ... do
3:      $N_i \leftarrow$  Receive availability info from cloud provider
4:      $(D_i, P_i) \leftarrow$  AdjustParallelConfiguration( $N_i$ )
5:      $S_i \leftarrow$  GetMigrationStrategy( $((D_{i-1}, P_{i-1}), (D_i, P_i))$ )
6:     Send migration strategy  $S_i$  to all ParcaeAgents
7:      $N_{i+1}, \dots, N_{i+I} \leftarrow$  AvailPredictor( $N_i, H+1, \dots, N_i$ )
8:      $(D_{i+1}, P_{i+1}) \leftarrow$  LiveputOpt( $((D_i, P_i), N_i, \dots, N_{i+I})$ )
9:   if job completes then
10:    break

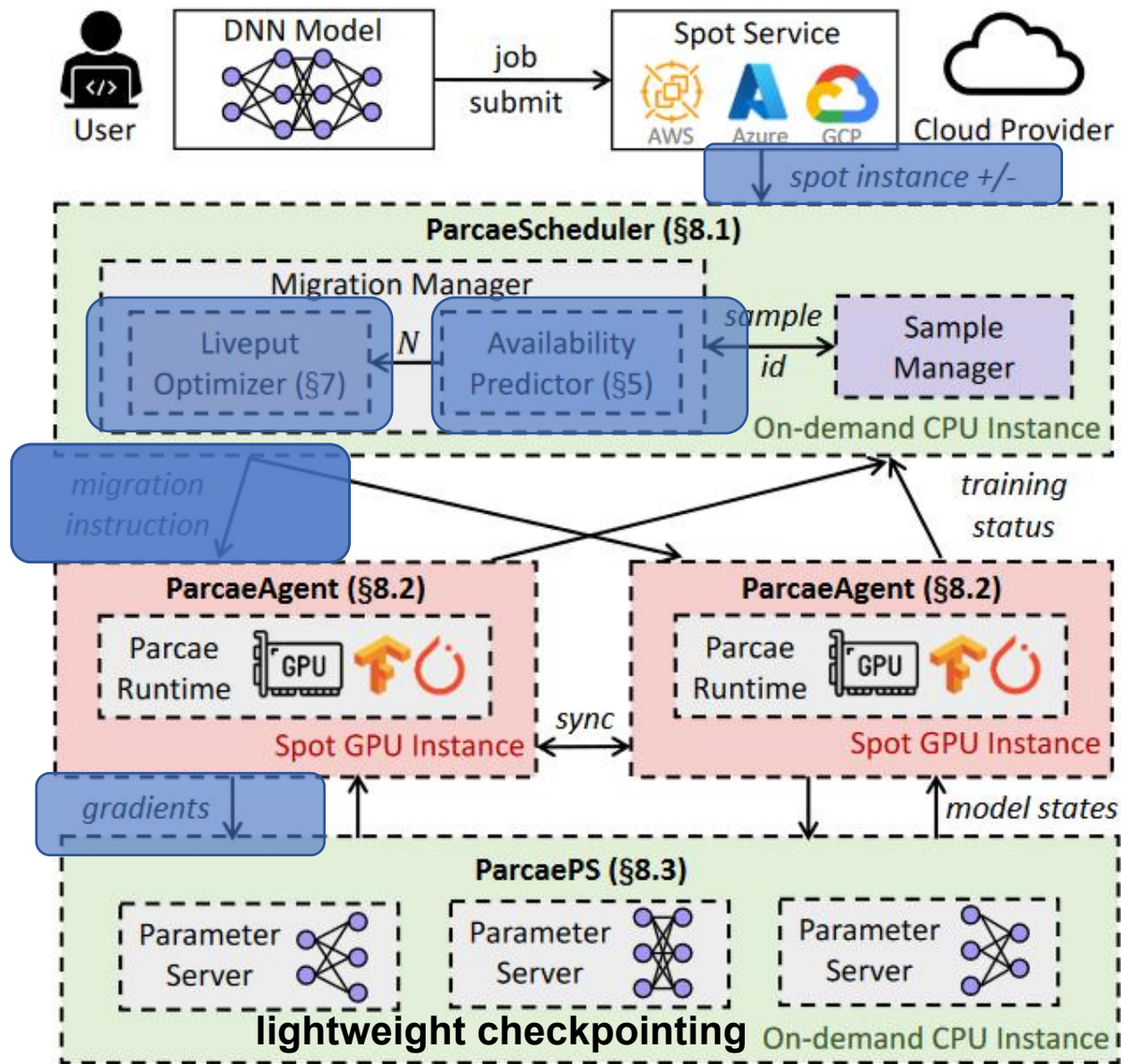
```

▷ ParcaeAgent

```

11: function PARCAERUNTIME(model, batch_size)
12:   while job does not complete do
13:     Receive migration instruction  $m$  from ParcaeScheduler
14:     Apply migration instruction  $m$  if  $m$  is not empty
15:      $X, Y \leftarrow$  DataLoader(batch_size)
16:     Train(model,  $X, Y$ )

```



Evaluation

- **DNN Models:**

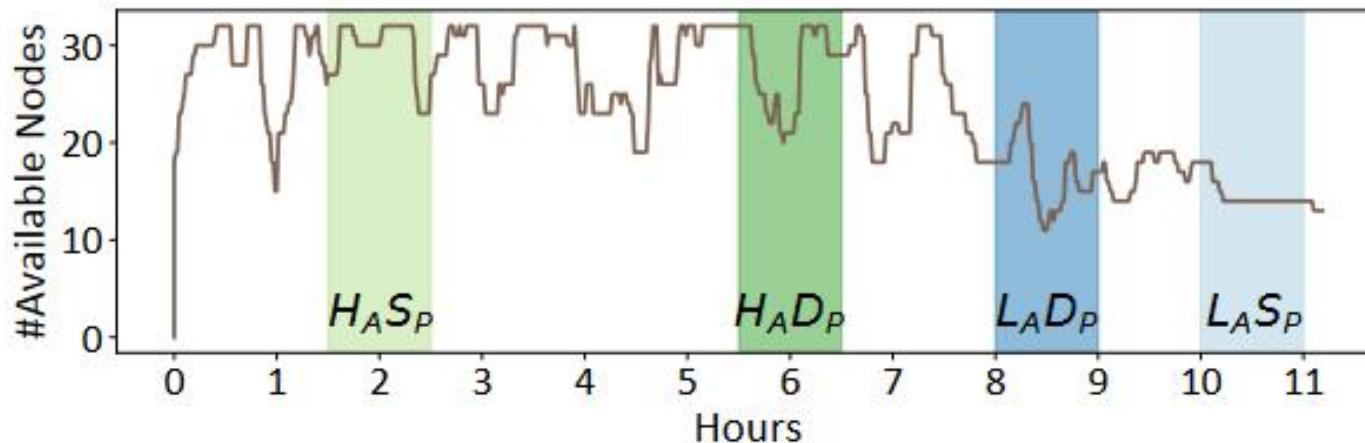
- Computer Vision (CV) Tasks:

- ResNet-152 and VGG-19
- Training Dataset: CIFAR-100

- Natural Language Processing (NLP) Tasks:

- BERT, GPT-2(1.5B), GPT-3(6.7B)
- Evaluation Dataset: WikiText-2

- Real Spot Instance Traces: 12-hour trace on a **32-instance cluster** (p3.2xlarge, AWS)



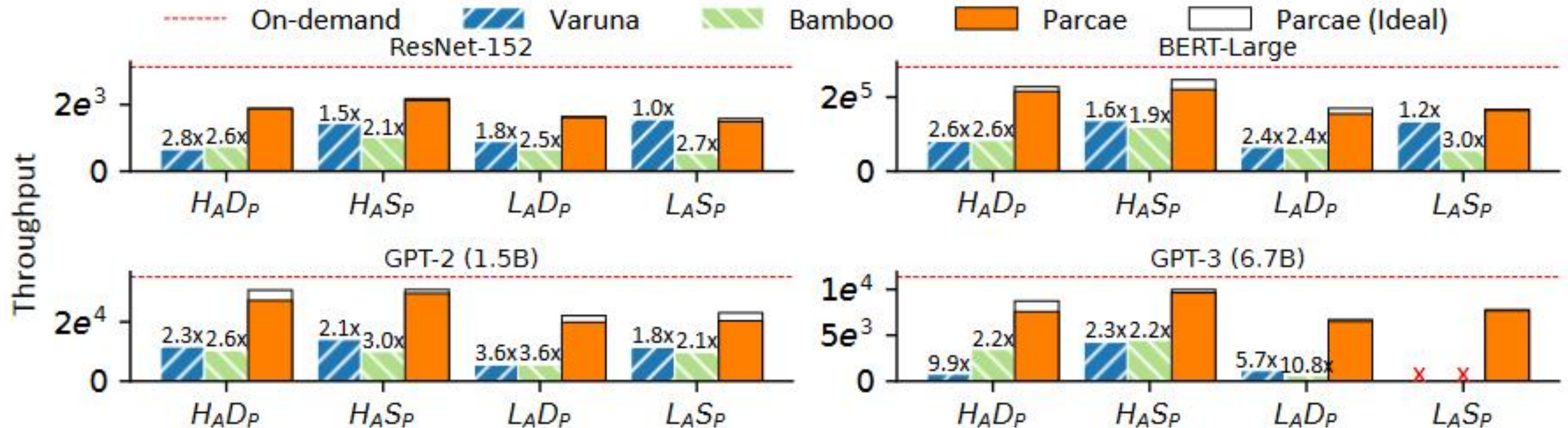
- Replayed on **32 on-demand V100-16GB GPUs** to simulate spot-instance clusters for consistent evaluation.

Table 3: Overview of the five DNNs evaluated.

Model	mini-batch	micro-batch	Dataset
ResNet-152 [18]	2048	32	CIFAR-100 [24]
VGG-19 [45]	2048	32	CIFAR-100 [24]
BERT-Large [14]	1024	8	WikiText-2 [28]
GPT-2 (1.5B) [38]	128	1	WikiText-2 [28]
GPT-3 (6.7B) [12]	64	1	WikiText-2 [28]

- High Availability (HA) : > 70% available nodes
- Low Availability (LA) : < 70%
- Dense (DP): ~20 preemption/allocation events
- Sparse (SP): Few events

Evaluation



- Parcae looks ahead 12 intervals based on the **availability predictor**.
- Parcae (Ideal) looks ahead 12 intervals based on **truth traces**.

Parcae delivers an overall of 2.59× higher throughput than Varuna and 3.0× than Bamboo.

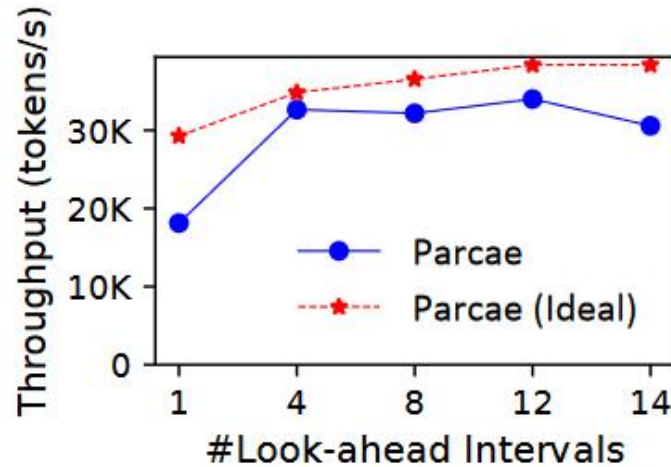
Evaluation

Model	Trace	On-Demand	Varuna	Bamboo	Parcae
ResNet	H_{ADP}	8.68 (2.3×)	10.86 (2.8×)	9.77 (2.6×)	3.81 (1×)
	H_{ASP}	8.68 (2.4×)	5.32 (1.5×)	7.61 (2.1×)	3.62 (1×)
	L_{ADP}	8.68 (3.2×)	4.89 (1.8×)	6.72 (2.5×)	2.71 (1×)
	L_{ASP}	8.68 (3.4×)	2.43 (1.0×)	6.96 (2.7×)	2.54 (1×)
VGG	H_{ADP}	12.43 (2.7×)	12.10 (2.6×)	12.11 (2.6×)	4.62 (1×)
	H_{ASP}	12.43 (2.7×)	6.52 (1.4×)	13.12 (2.8×)	4.66 (1×)
	L_{ADP}	12.43 (3.4×)	5.43 (1.5×)	9.40 (2.6×)	3.66 (1×)
	L_{ASP}	12.43 (4.0×)	3.37 (1.1×)	8.88 (2.9×)	3.11 (1×)
BERT	H_{ADP}	0.10 (2.9×)	0.09 (2.6×)	0.09 (2.6×)	0.03 (1×)
	H_{ASP}	0.10 (2.8×)	0.06 (1.6×)	0.06 (1.9×)	0.03 (1×)
	L_{ADP}	0.10 (3.4×)	0.07 (2.4×)	0.07 (2.4×)	0.03 (1×)
	L_{ASP}	0.10 (4.2×)	0.03 (1.2×)	0.07 (3.0×)	0.02 (1×)
GPT-2	H_{ADP}	0.62 (2.9×)	0.49 (2.3×)	0.55 (2.6×)	0.21 (1×)
	H_{ASP}	0.62 (3.0×)	0.44 (2.1×)	0.62 (3.0×)	0.21 (1×)
	L_{ADP}	0.62 (3.5×)	0.63 (3.6×)	0.64 (3.6×)	0.18 (1×)
	L_{ASP}	0.62 (4.1×)	0.27 (1.8×)	0.31 (2.1×)	0.15 (1×)
GPT-3	H_{ADP}	2.39 (2.5×)	9.35 (9.9×)	2.07 (2.2×)	0.94 (1×)
	H_{ASP}	2.39 (3.0×)	1.81 (2.3×)	1.74 (2.2×)	0.80 (1×)
	L_{ADP}	2.39 (3.6×)	3.81 (5.7×)	7.28 (10.8×)	0.67 (1×)
	L_{ASP}	2.39 (4.8×)	-	-	0.49 (1×)

- Compared with on-demand
- Parcae is **3.24× cheaper**

Evaluation

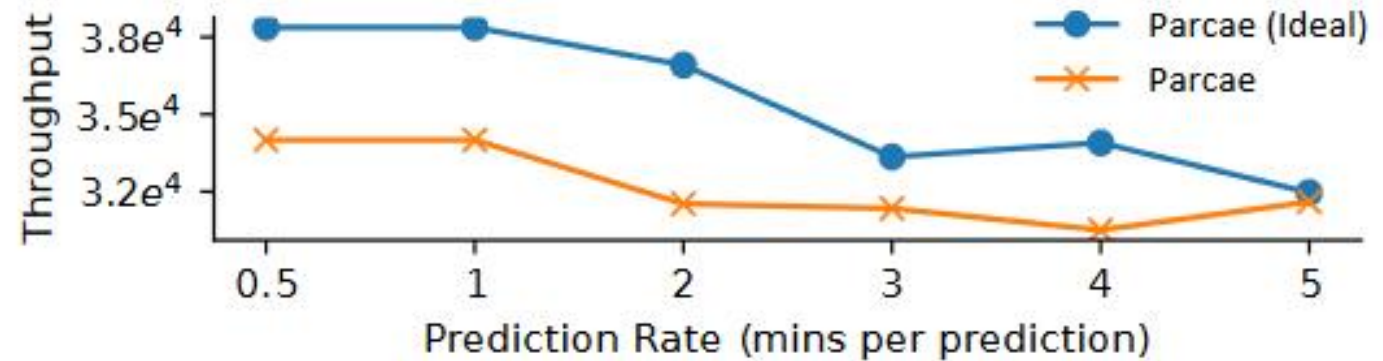
Look-ahead Interval Length



Look-ahead Window (Optimal = 12 Intervals)

- Effect: Improves decision quality & throughput
- Principle: Longer look-ahead better adapts to dynamic resources

Prediction Rate

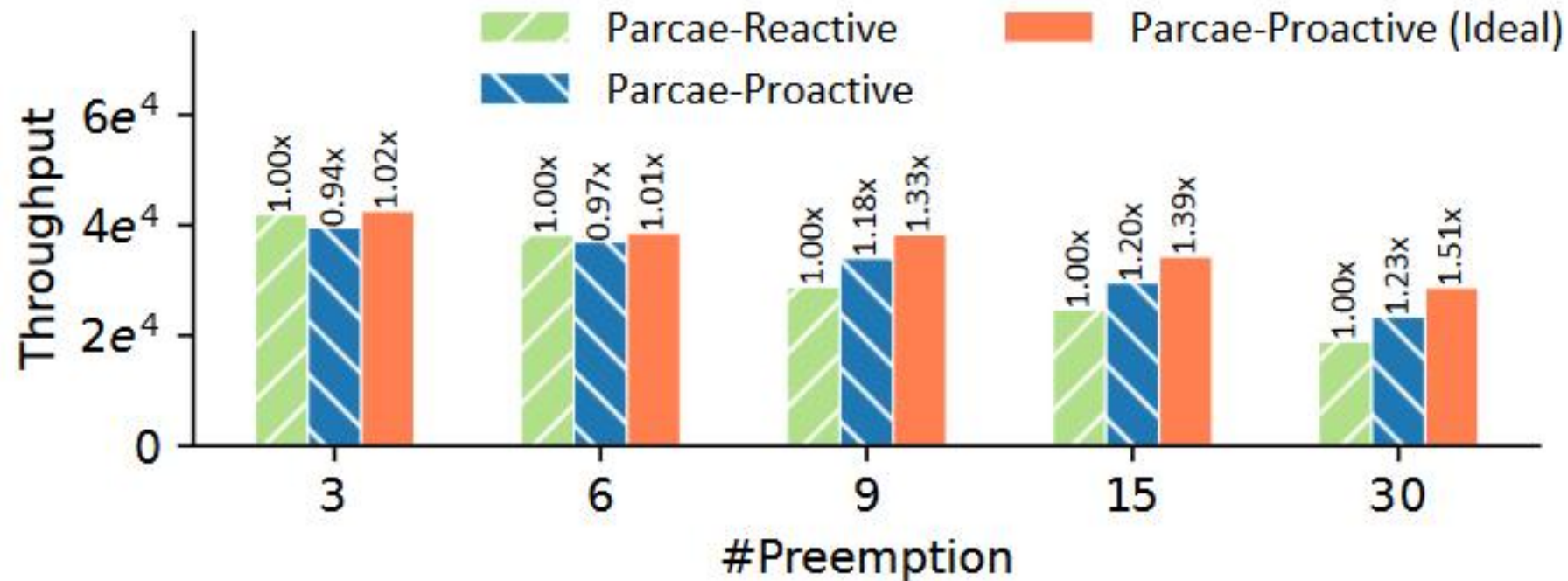


- Higher throughput with higher prediction rates
- Allowing **minute-level** prediction and optimization.

Evaluation

- ***Proactive v.s. Reactive***

- Parcae-Proactive (liveput optimization)
- Parcae-Reactive (throughput optimization)



Parcae's proactive strategy can be more effective for scenarios with more frequent preemptions.

Thinking

- **文章有什么问题，基于这个 paper 还能做什么？**

预测准确性不足：

- 文中采用轻量级的统计方法（ARIMA）来预测未来的实例可用性。虽然这种方法在实验中表现较好，但在面对云平台复杂、多变的抢占环境时，其预测准确性可能会受限。
- 抢占映射假设所有实例具有相同抢占概率，这一简化假设在实际中可能不够准确。（融入更多外部信息（例如实时价格、用户行为等），使预测模型更全面）

系统调优与参数选择：

- Liveput 优化中的许多超参数（如预测窗口长度、预测频率等）在实验中经过了仔细的调优，对于不同的 DNN 模型或不同云平台，适应性策略可能需要一些通用的机制或自适应学习能力。

Thinking

- **Paper 的 idea 能不能应用在自己的工作上面？**

- 文中提出了一种在动态环境下主动预测未来资源的可用性并协调不同并行配置的思路，这种方法适用于大规模的分布式训练中。
- 在涉及多层次并行（如数据、流水线等）的应用中，Parcae 的动态调度算法具有良好的扩展性。

- **这个 paper 能不能泛化？**

- 在动态资源环境中需要权衡性能的场景，都可以引入类似liveput 的指标。
- 迁移策略对于不同的任务（比如任务间依赖关系更加复杂）需要重新设计以满足特定数据依赖的要求，但主动预测与动态调度框架依然可以适用。



Q & A

Presenter: Yujie Chen
2024.4.18