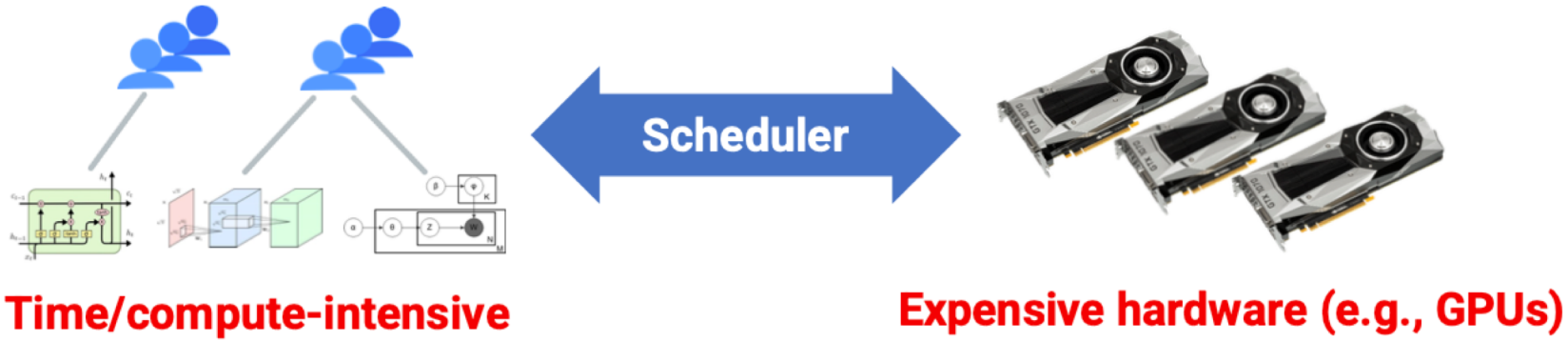
弹性调度综述

2022/5/17

Introduction

Scheduler : allocate resources to jobs s.t.

- minimize training time
- maximize cluster utilization
- ensure fairness



Summary

Paper	Conference	Affiliation	Open Source
Resource Elasticity...	MLSys 2020	Princeton/Google	-
Pollux	OSDI 2021	Petuum/CMU/Berkeley/MBZUAI	adaptdl
DeepPool	MLSys 2022	MIT	deeppool
Pathways	MLSys 2022	Google	-

Resource Elasticity in Distributed Deep Learning

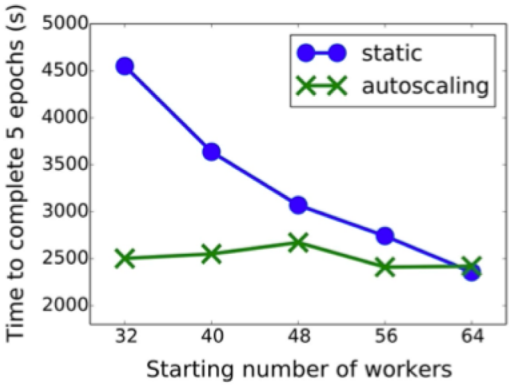
Resource Elasticity in Distributed Deep Learning

Andrew Or et al. Princeton University, Google AI

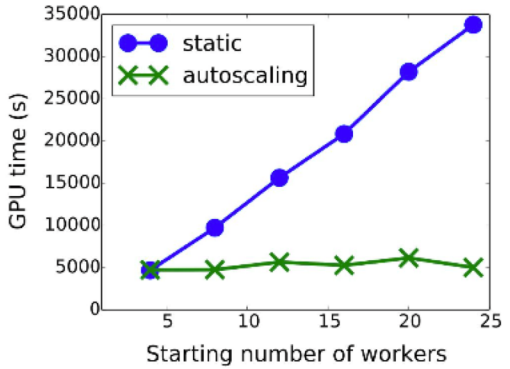
Autoscaling to dynamically search for a resource efficient cluster

Leads to **shorter job completion times** and **lower costs**

up to 45% reduction



up to 85.1% reduction in GPU time



Resource Elasticity in Distributed Deep Learning

Dataset

Model

Hardware

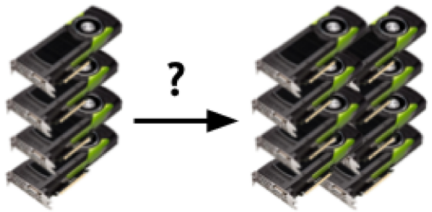
16 GPUs	2200 images/sec	✓
32 GPUs	4000 images/sec	✗
64 GPUs	5000 images/sec	

Users rely on manual trial-and-error process to find resource efficient cluster size

Resource Elasticity in Distributed Deep Learning

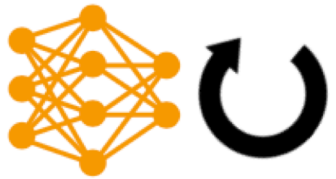
Cumbersome: difficult to estimate scaling behavior

Diverse hardware topologies, communication algorithm etc.



Time-consuming: each trial restarts entire program

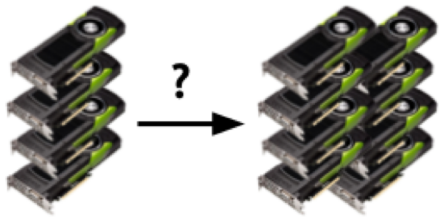
Need to reload libraries, rebuild model, prepare input pipeline etc.



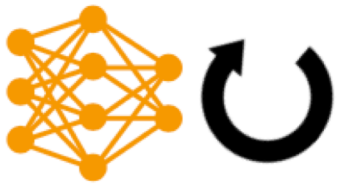
Can take minutes of device idle time

Static allocation: vulnerable to stragglers

Resource Elasticity in Distributed Deep Learning



Cumbersome



Time-consuming

Static allocation

Today, users often under- or over-allocate resources

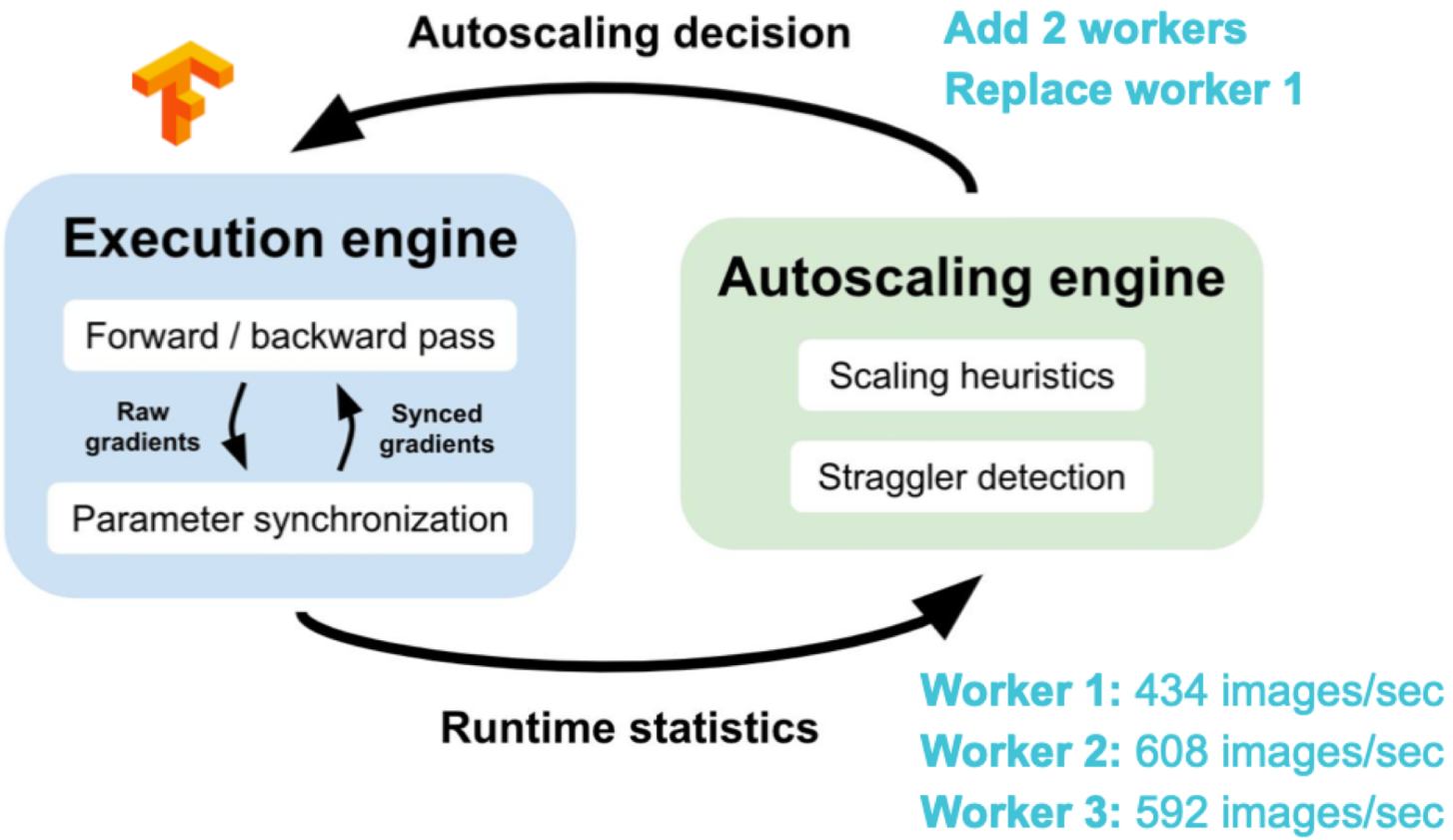
Resource Elasticity in Distributed Deep Learning

Why is resource elasticity not adopted yet?

- Lack of applicable scaling heuristics
- Existing frameworks assume static allocation
- How to scale the batch size ?

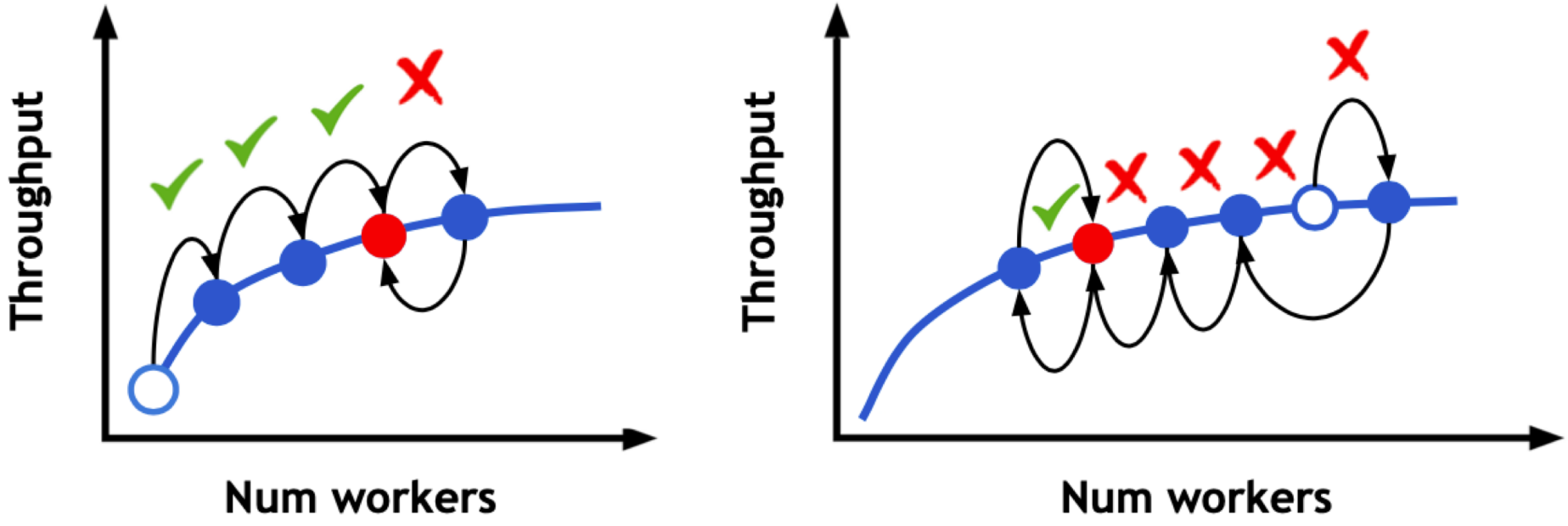
Resource Elasticity in Distributed Deep Learning

Autoscaling engine



Resource Elasticity in Distributed Deep Learning

Find the latest point at which the **scaling condition** passes ✓



Resource Elasticity in Distributed Deep Learning

- Implementation: Horovod
- Once we detect a straggler, replace it using the same mechanisms
- Finding an optimal batch size for arbitrary workloads is an open problem

Pollux

Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning

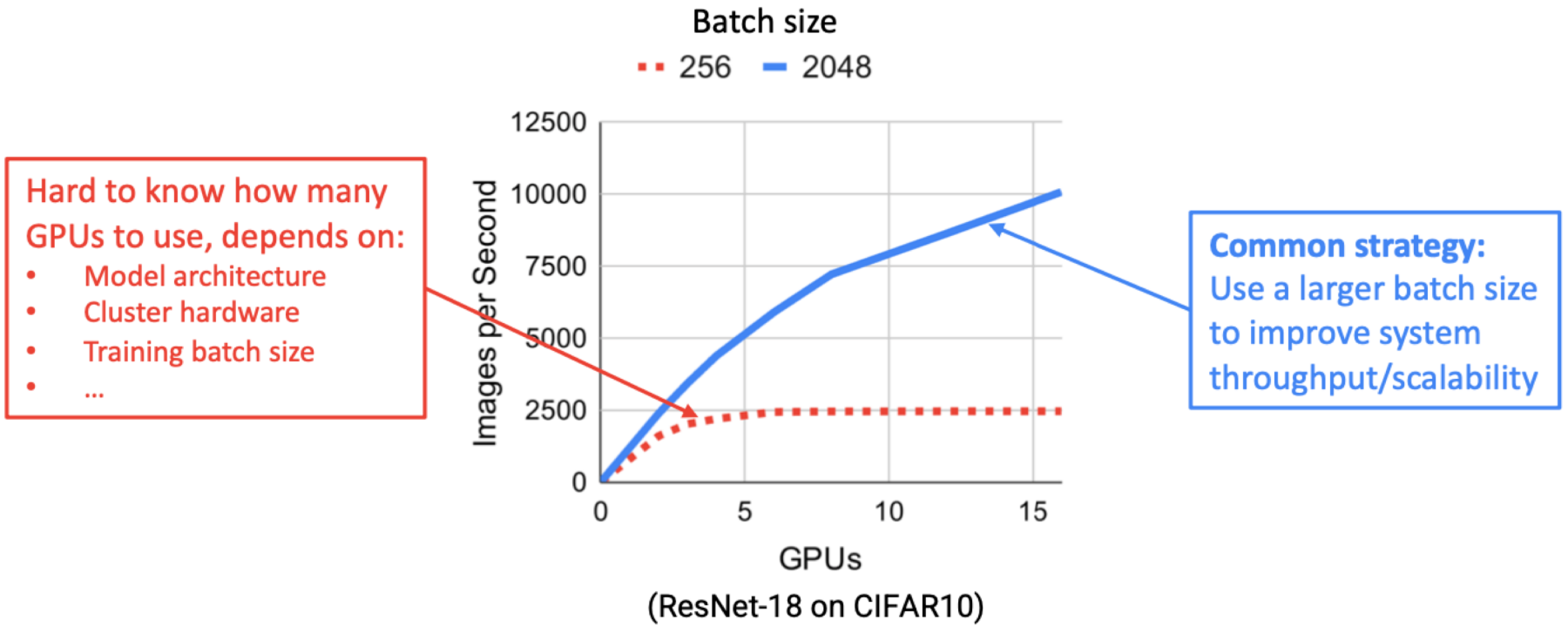
Aurick Qiao et al. Petuum, Inc, CMU, Berkeley, MBZUAI.

- Pollux co-optimizes both cluster-wide and per-job parameters for DL training
- Pollux introduces goodput, a measure of training performance that combines system throughput and statistical efficiency
- Pollux improves average training time in shared clusters by 37-50%, even against unrealistically strong baselines

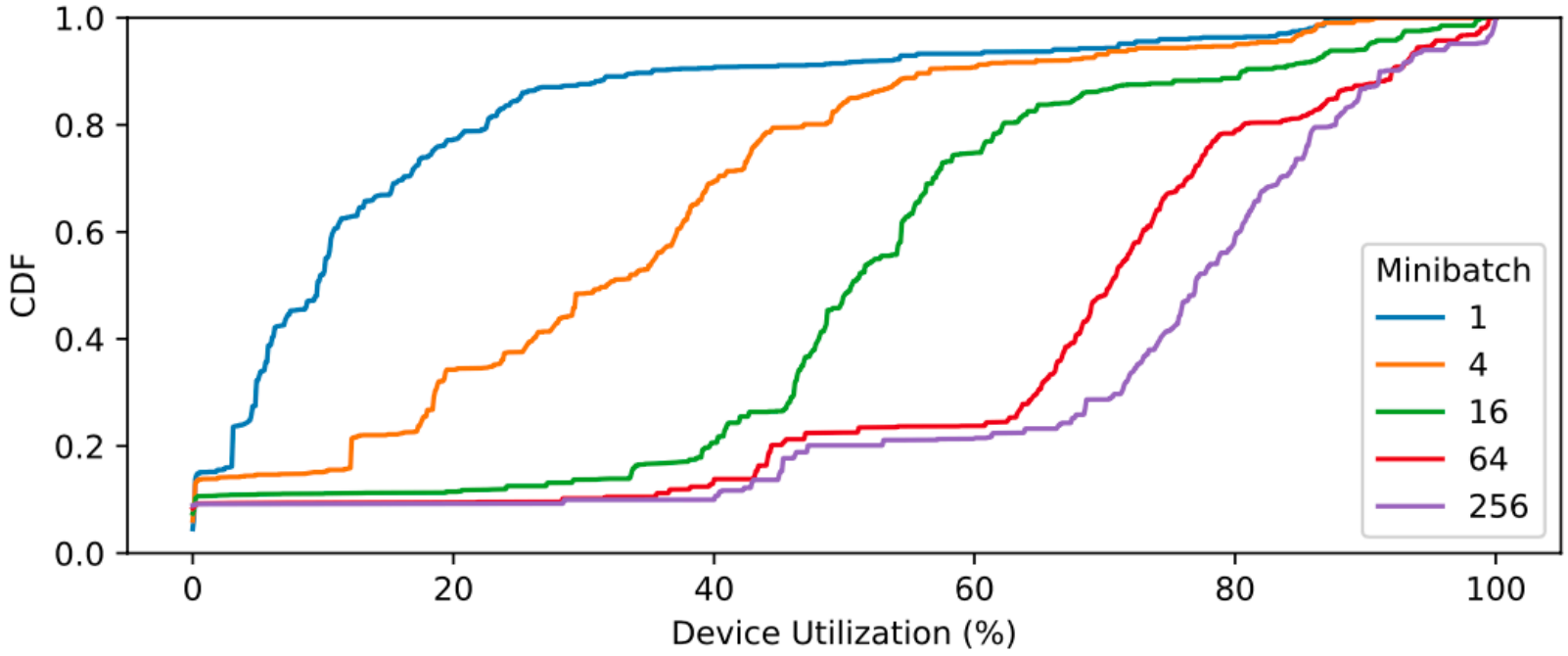
Pollux

DL training scales **sub-linearly,**

too many GPUs do not increase system throughput!

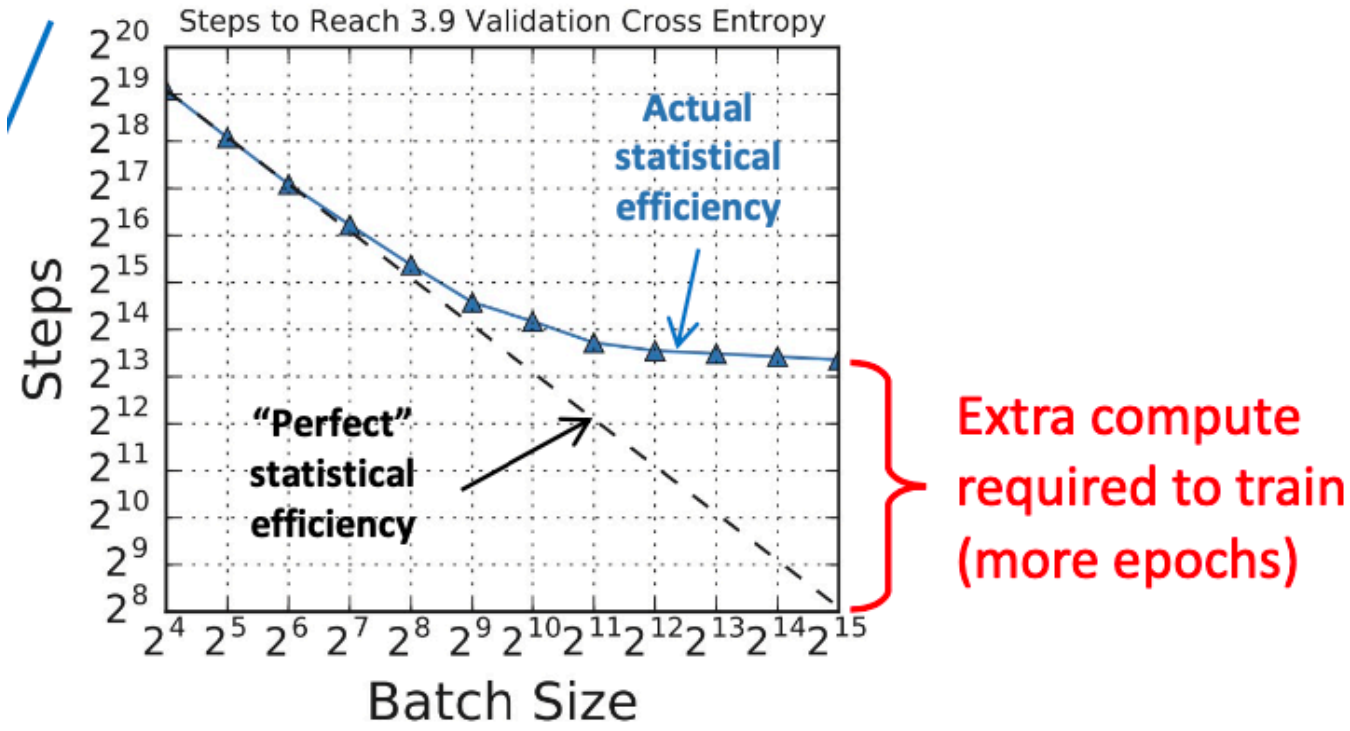


Pollux


1.0
0.8
0.6
0.4
0.2
0.0
CDF
0
20
40
60
80
100
Device Utilization (%)
Minibatch
1
4
16
64
256

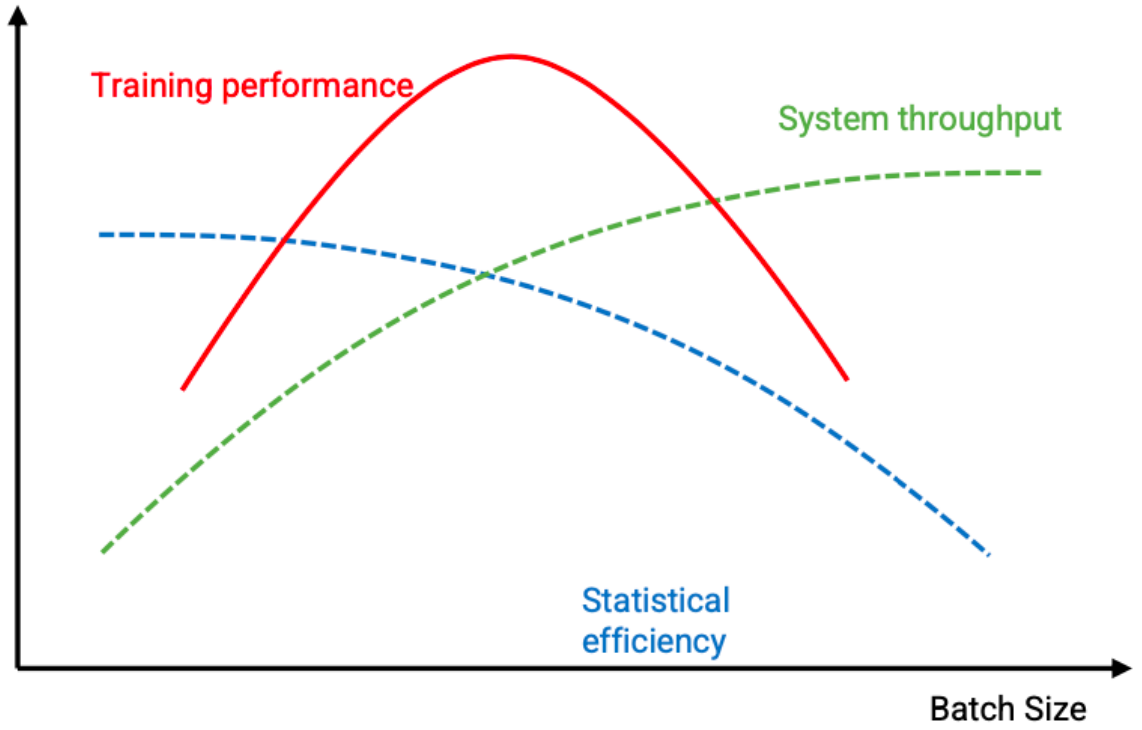
Pollux

- Increasing the batch size decreases the **statistical efficiency** of DL training



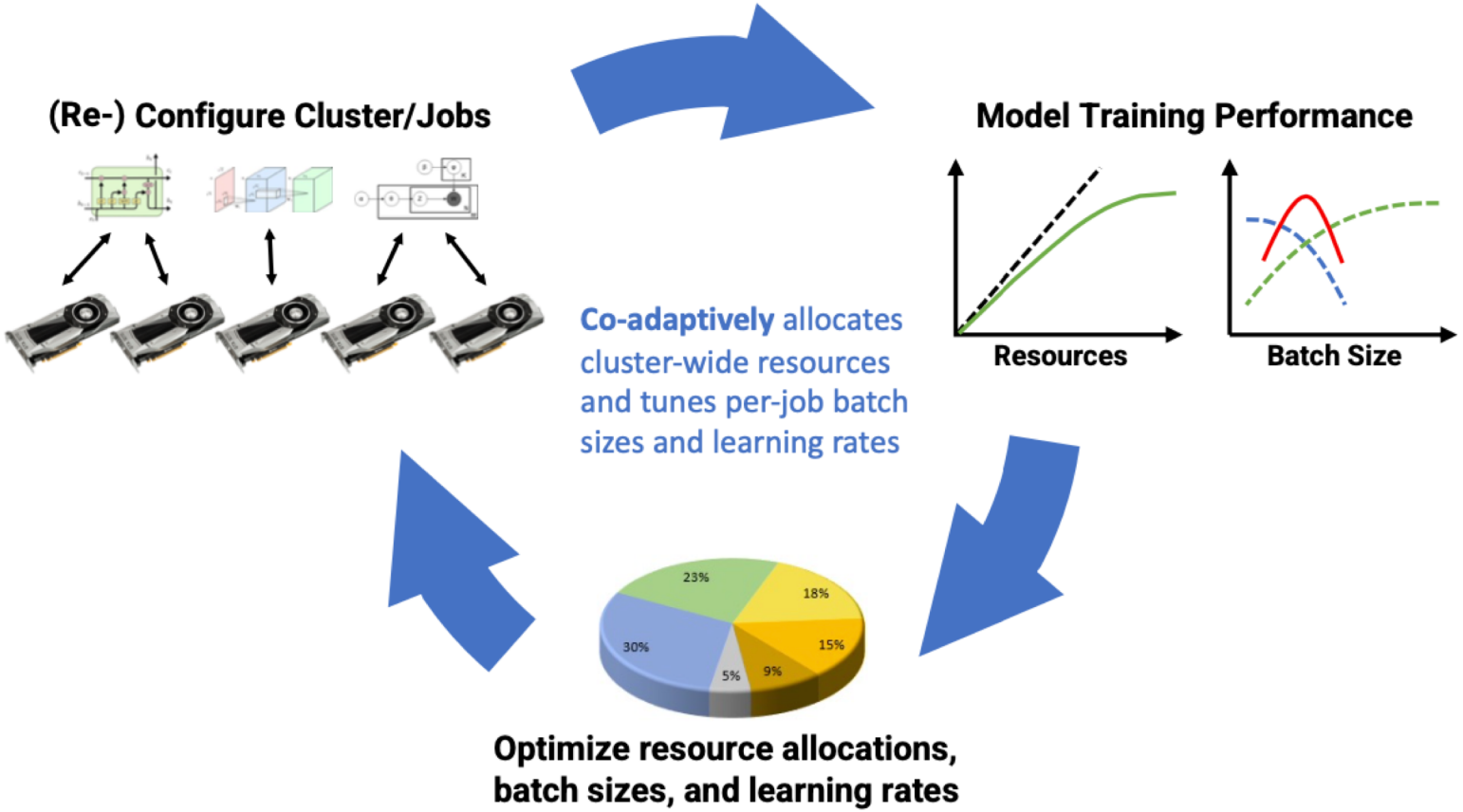
Pollux

- Statistical efficiency increases during training



Pollux

- Cluster Scheduler



Pollux

- Pollux optimizes for a new measure of DL training performance *goodput*

$$\text{GOODPUT}_t(a, m, s) = \underbrace{\text{THROUGHPUT}(a, m, s)}_{\textbf{\textit{System} throughput}} \times \underbrace{\text{EFFICIENCY}_t(M)}_{\textbf{\textit{Statistical} efficiency}}$$

(training examples / second)

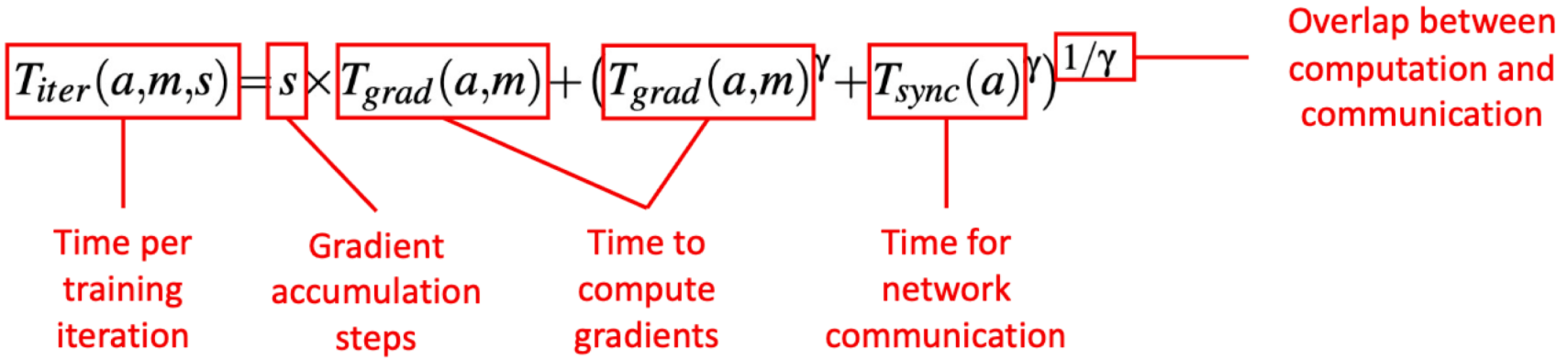
(progress / training example)

Automatically determined by Pollux for each job during training

- a: Allocation vector, a_n = #GPUs on node n
- m: Per-GPU batch size
- s: Gradient accumulation steps (enables total batch sizes larger than GPU memory limit)
- M: Total batch size, $M = |a| \times m \times s$

Pollux

- Modeling System Throughput



- Modeling Statistical Efficiency

$$\texttt{EFFICIENCY}_t(M) = \frac{\varphi_t + M_0}{\varphi_t + M}$$

User provides a static baseline batch size M_0

EFFICIENCY of batch size M is measured relative to M_0

Pollux

- Optimizes Goodput for Each DL Job

Given an allocation of GPUs a, solve

$$m^*, s^* = \underset{m,\,s}{\operatorname{argmax}} \, \mathrm{GOODPUT}_t(a, m, s)$$

- Optimizes Cluster-Wide Allocations

Pollux finds an *allocation matrix* A

$$\mathrm{FITNESS}_p(A) = \left(\frac{1}{J} \sum_{j=1}^{J} \mathrm{SPEEDUP}_j(A_j)^p \right)^{1/p}$$

$$\mathrm{SPEEDUP}_j(A_j) = \frac{\max_{m,s} \mathrm{GOODPUT}_j(A_j, m, s)}{\max_{m,s} \mathrm{GOODPUT}_j(a_f, m, s)}$$

Pollux

Results

16 nodes w/ 4 GPUs each (Nvidia T4)

160 DL jobs submitted over 8 hours

Policy	Job Completion Time		Makespan
	Average	99%tile	
Pollux ($p=-1$)	**0.76h**	11h	**16h**
Optimus+Oracle+TunedJobs	1.5h	15h	20h
Tiresias+TunedJobs	1.2h	15h	24h

Uses expert-configured jobs.
Can elastically adapt resources, but not batch size/learning rate.
(Peng et al. 2018)

Uses expert-configured jobs.
Can pause/resume jobs based on their GPU-time metrics.
(Gu et al. 2019)

37-50% faster average training time

More realistic job configs:
>70% faster average training time

DeepPool

Efficient Strong Scaling Through Burst Parallel Training

Seo Jin Park et al. MIT

- *burst parallelism* allocates large numbers of GPUs to foreground jobs *in bursts* to exploit the unevenness in parallelism across layers
- *GPU multiplexing* prioritizes throughput for foreground jobs, while packing in background jobs to reclaim underutilized GPU resources
- deliver a 1.2 – 2.3× improvement in total cluster throughput over standard data parallelism

DeepPool

Weak Scaling : increase the global batch size correspondingly

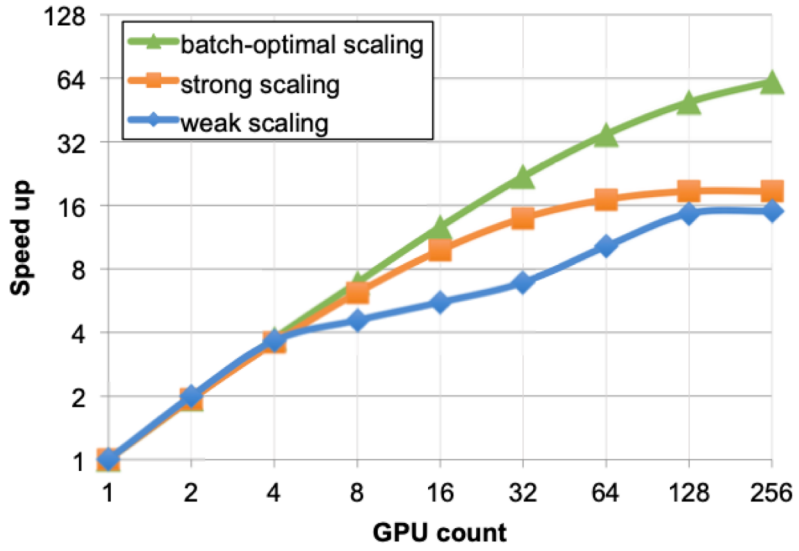
- improves training *throughput*
- degrade the sample efficiency

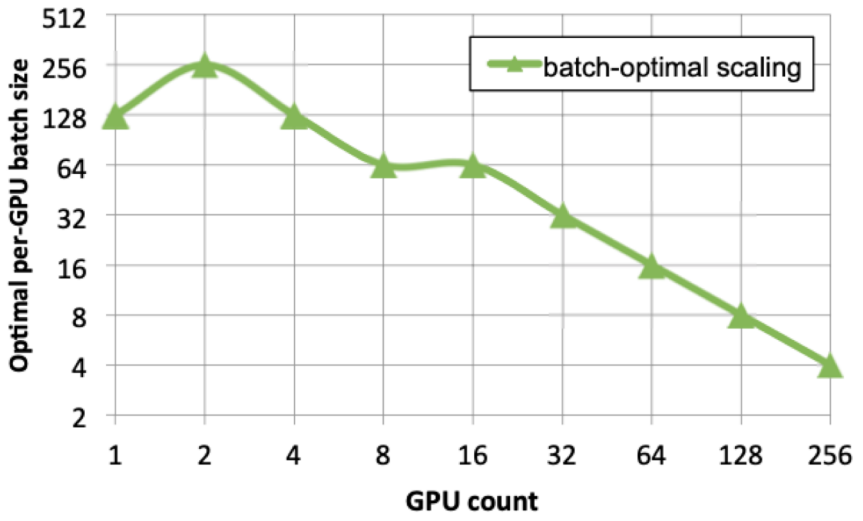
Strong Scaling : keep the global batch size fixed

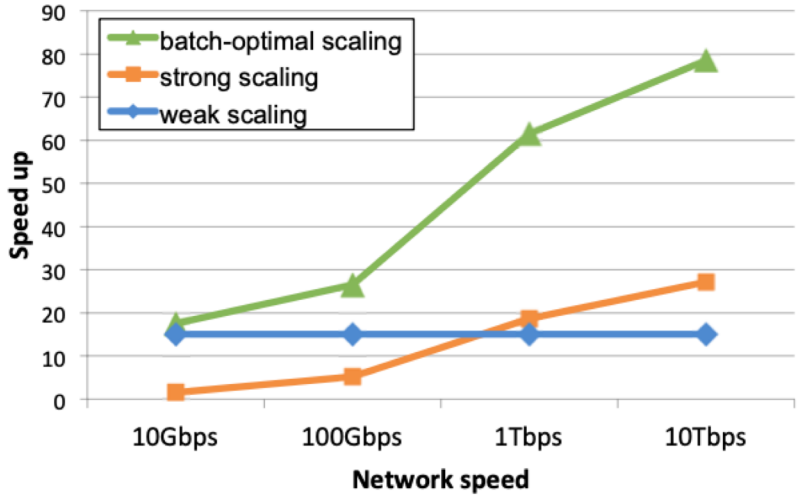
- bottlenecked by communication bandwidth between GPUs
- difficult to use cluster resources efficiently

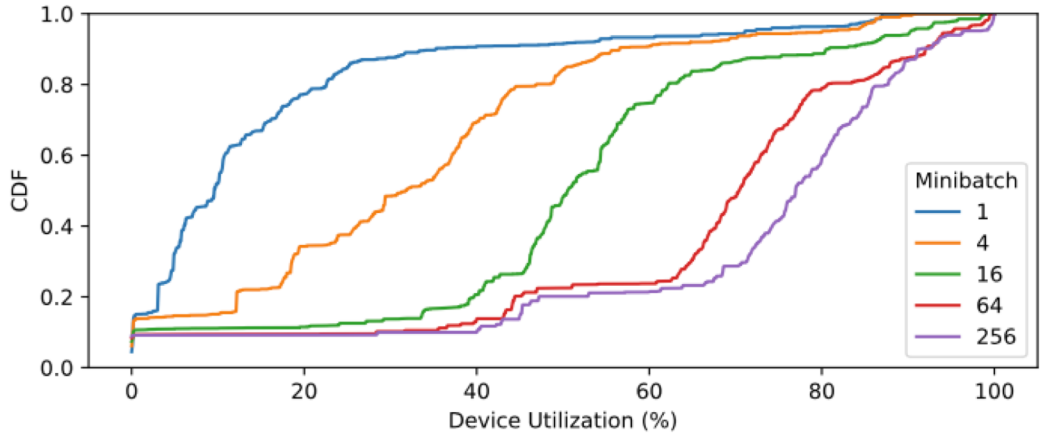
Batch-optimal Scaling : best time to accuracy

DeepPool


batch-optimal scaling
strong scaling
weak scaling
Speed up
GPU count
128
64
32
16
8
4
2
1
1
2
4
8
16
32
64
128
256


batch-optimal scaling
Optimal per-GPU batch size
GPU count
512
256
128
64
32
16
8
4
2
1
2
4
8
16
32
64
128
256


batch-optimal scaling
strong scaling
weak scaling
Speed up
Network speed
90
80
70
60
50
40
30
20
10
0
10Gbps
100Gbps
1Tbps
10Tbps


CDF
Device Utilization (%)
Minibatch
1
4
16
64
256
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

DeepPool

Interesting findings from experiments

- Optimizing time-to-accuracy requires small per-GPU batches at large scale
- Strong scaling and small per-GPU batches are more effective with fast networks
- None of the approaches achieve perfect linear scaling

DeepPool

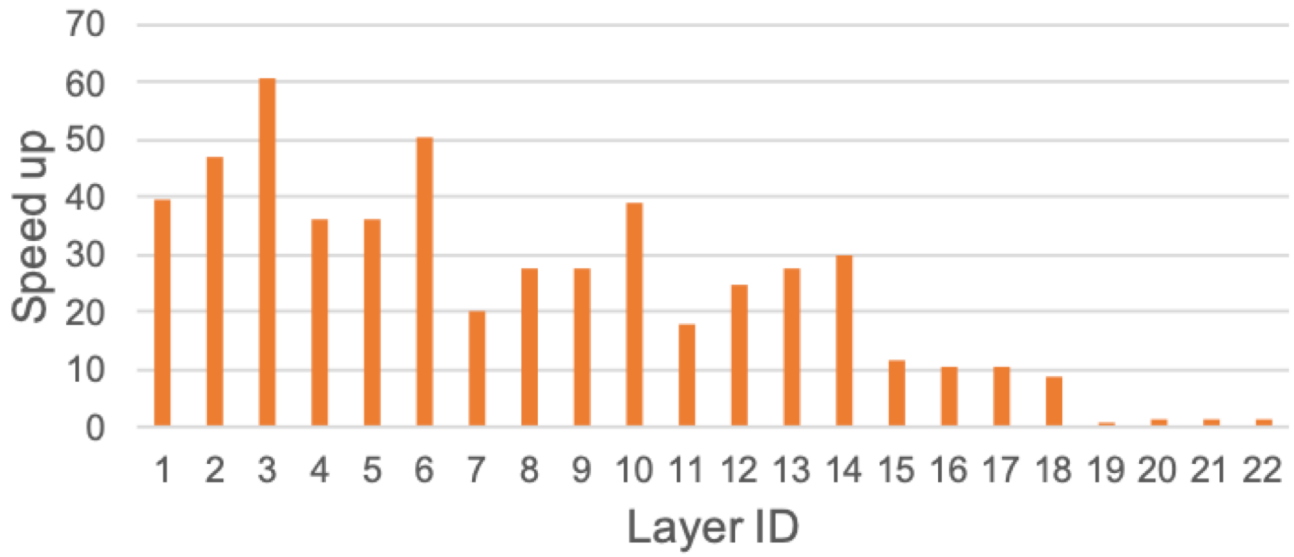


Figure 5. Heterogeneous scalability of layers in VGG16. Y-axis shows the speedup of each layer when it is strong scaled from 128 samples per iteration to 2 samples per iteration using 64 GPUs.

DeepPool

Burst parallel training to dynamically adjust the number of GPUs allocated to each layer, so that layers with less parallelism can use fewer GPUs.

GPU multiplexing techniques that allow multiple models to be trained on each GPU simultaneously

DeepPool

Burst parallel training planner (static graph)

- initially profiles each layer with different batch sizes
- decides the scaling of each layer

Cluster coordinator

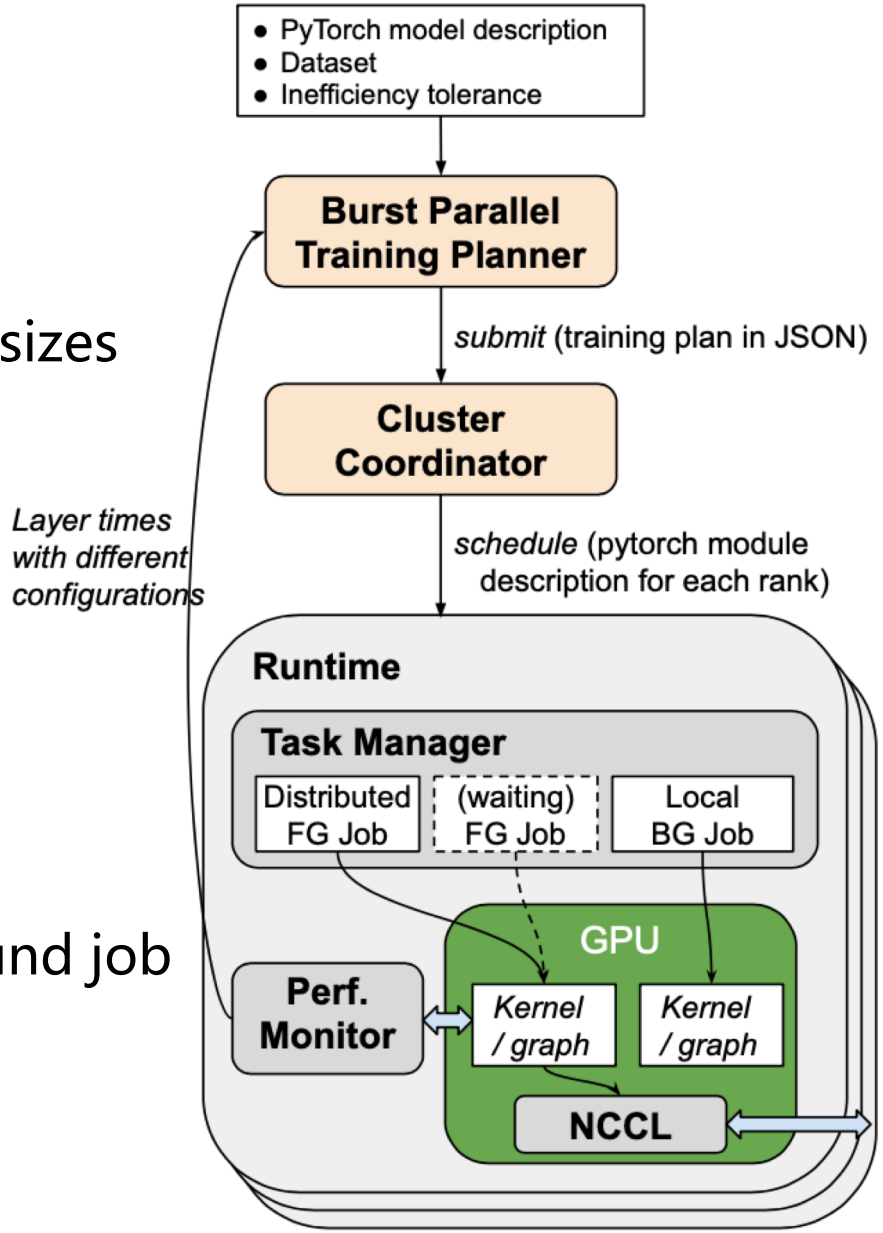
- places a new training job on a subset of GPUs

Task manager

- manage one foreground job and one background job

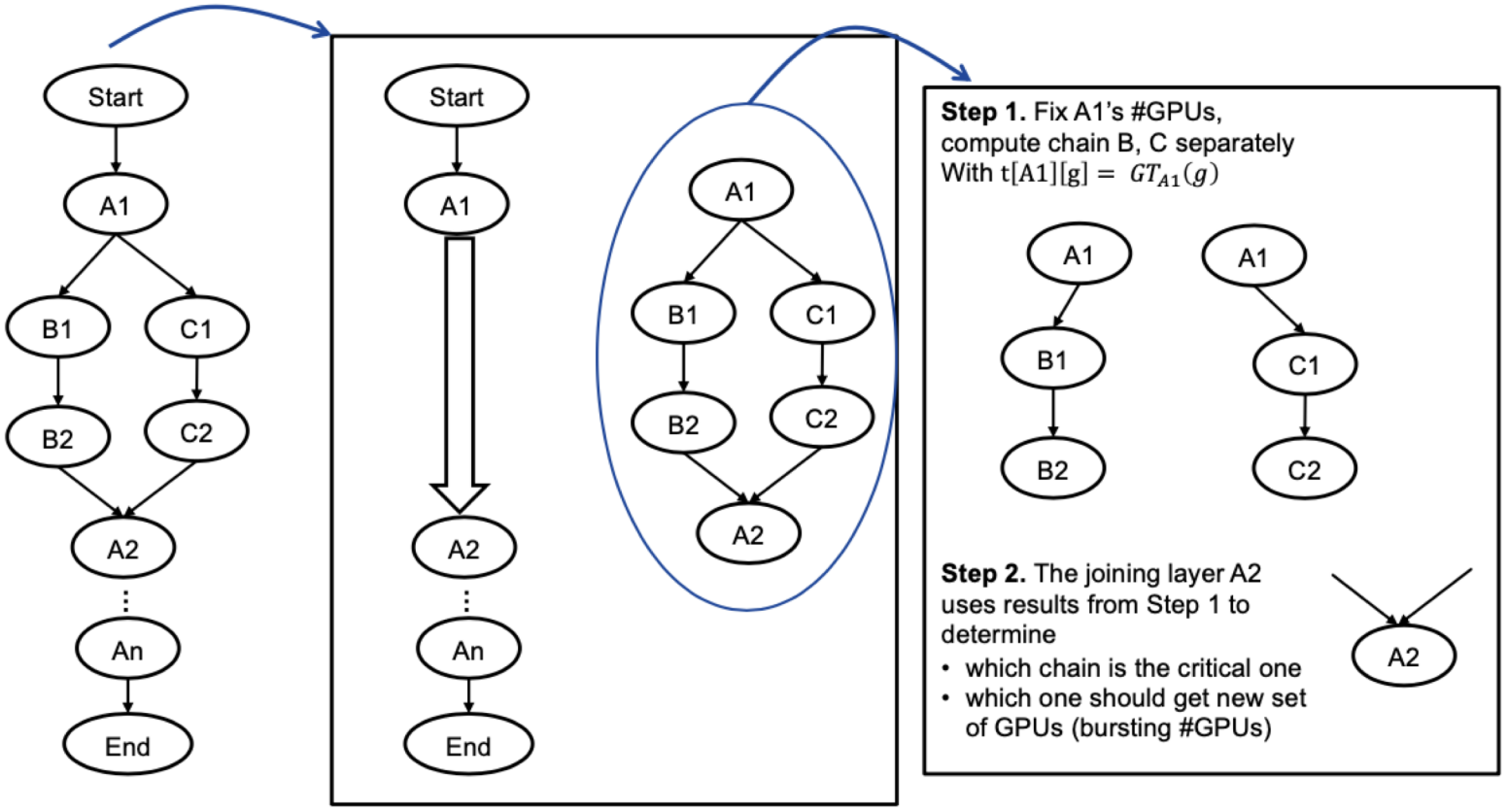
Performance monitor

- profile and feedback



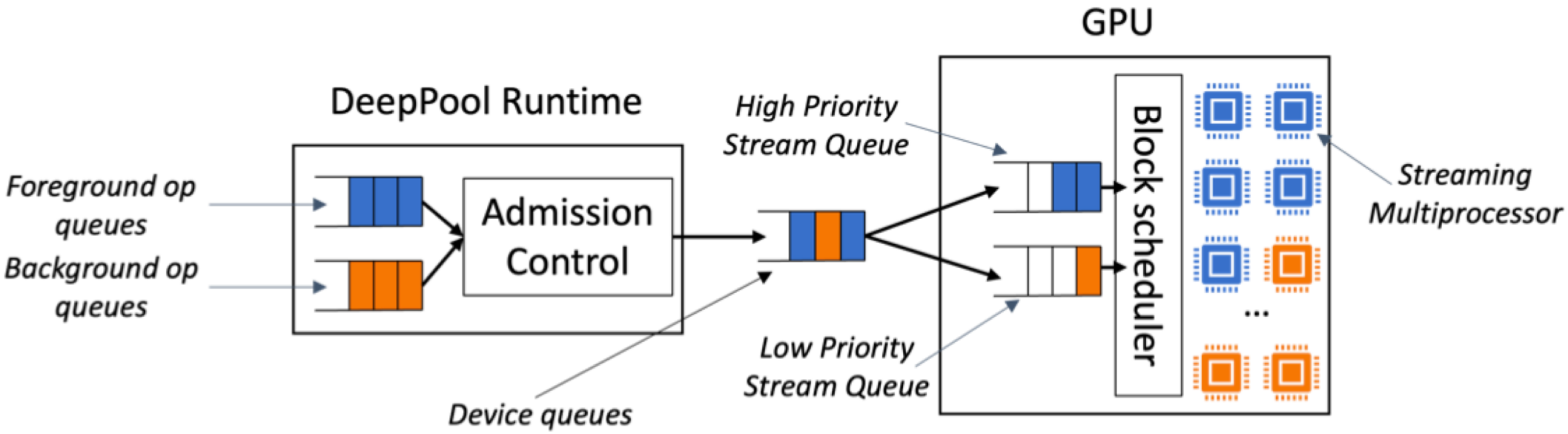
DeepPool

Burst parallel scheduling

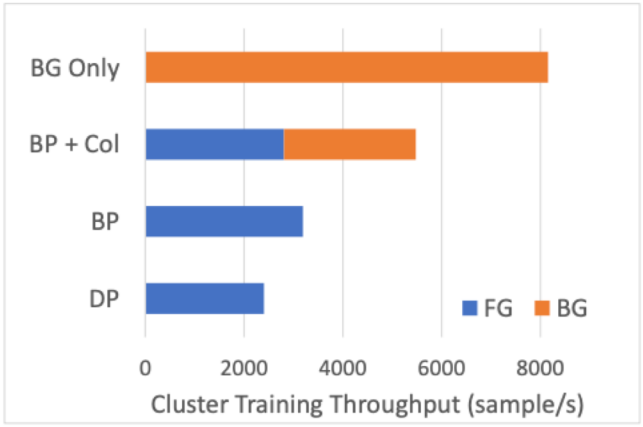


DeepPool

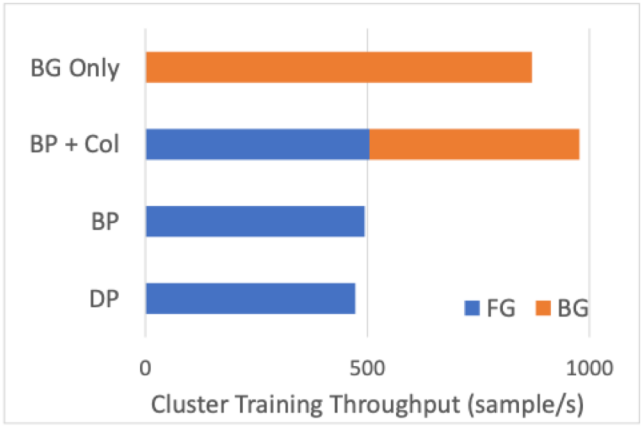
GPU multiplexing



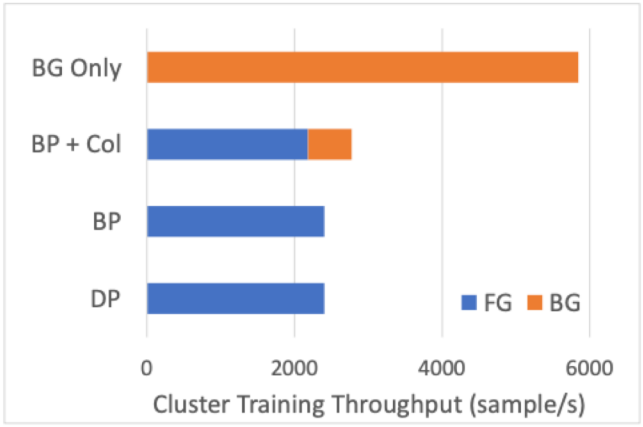
DeepPool



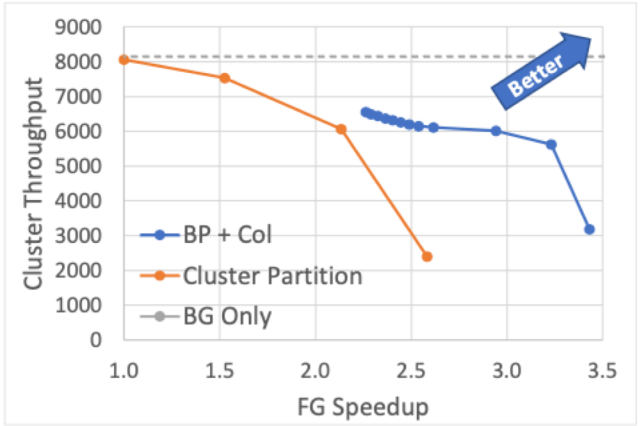
(a) VGG-16, scaling b=32



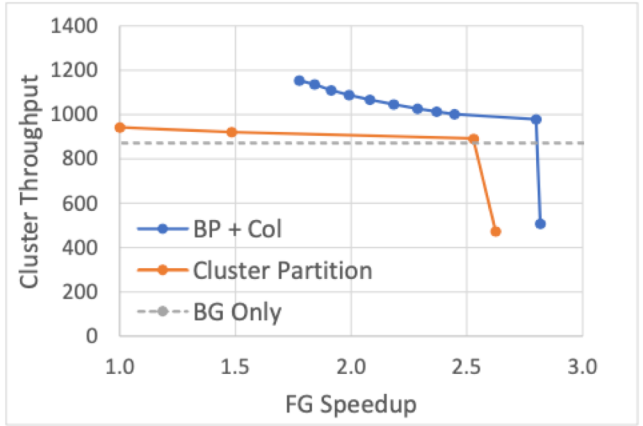
(b) WideResNet-101-2, scaling b=16



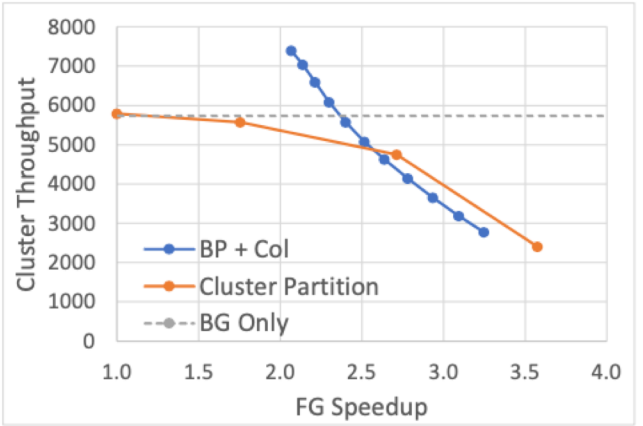
(c) InceptionV3, scaling b=32



(a) VGG-16, scaling b=32



(b) WideResNet-101-2, scaling b=16



(c) InceptionV3, scaling b=32

DeepPool

Limitations

- more flexible forms of model parallelism
- limit background jobs to a single GPU
- GPU hardware has issue of multiplexing efficiency

Pathways

Pathways: Asynchronous Distributed Dataflow for ML

Paul Barham et al. GOOGLE

Pathways : flexibility of single-controller + performance of multi-controllers

- single-controller model
- shared dataflow graph
- asynchronous operators
- gang-schedulers heterogeneous parallel computations

Pathways

SPMD : single program multiple data

MPMD : multiple program multiple data

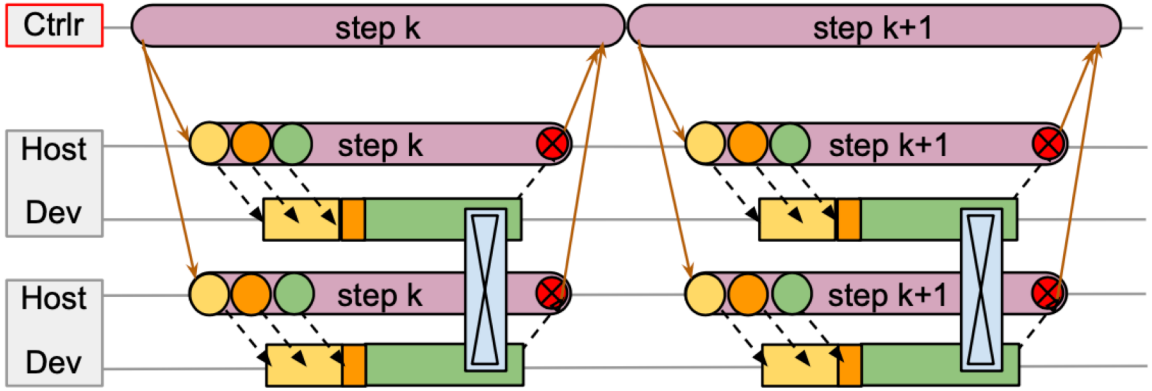
single-controller ~ MPMD

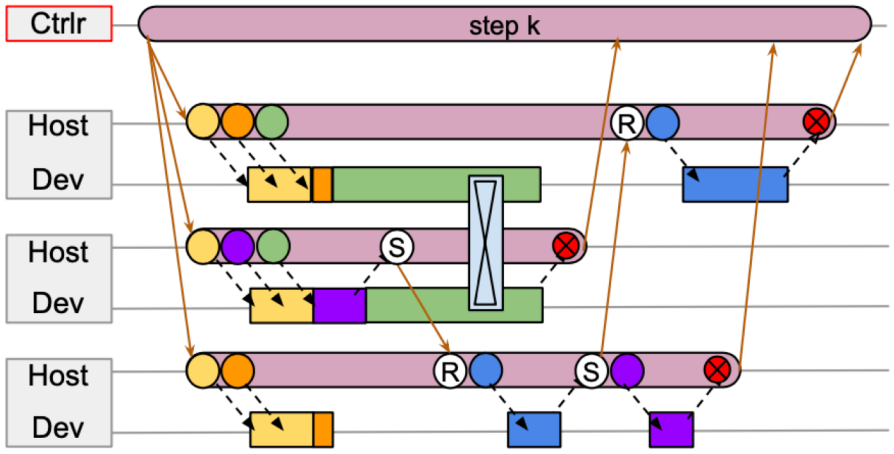
multi-controller ~ SPMD

Pathways

Single-controller

- general distributed dataflow model
- optimized in-graph control flow
- performance issue, implementation challenges (TensorFlow v1)

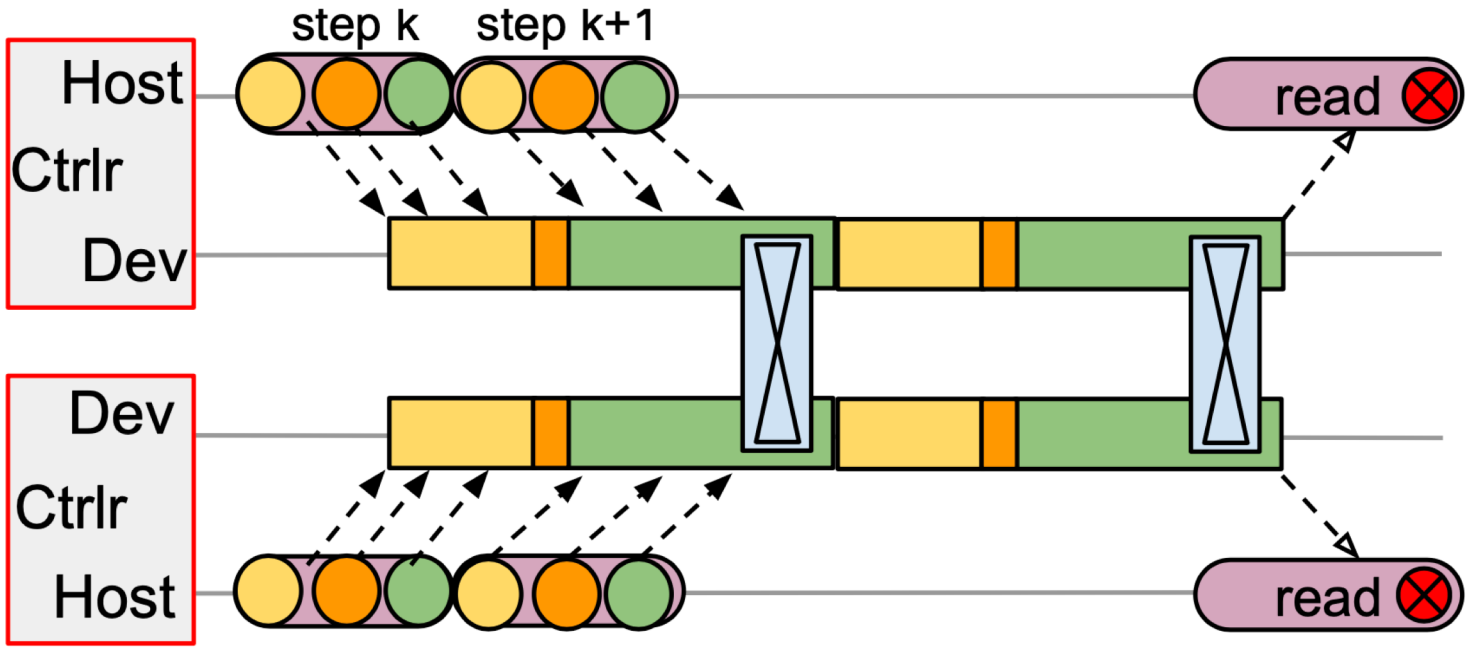




Pathways

Multi-controller ~ SPMD

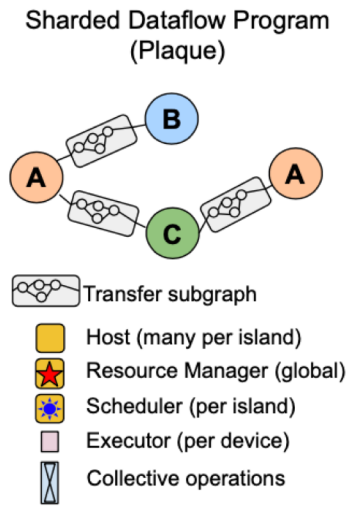
- low latency for dispatching, high performance
- less flexibility

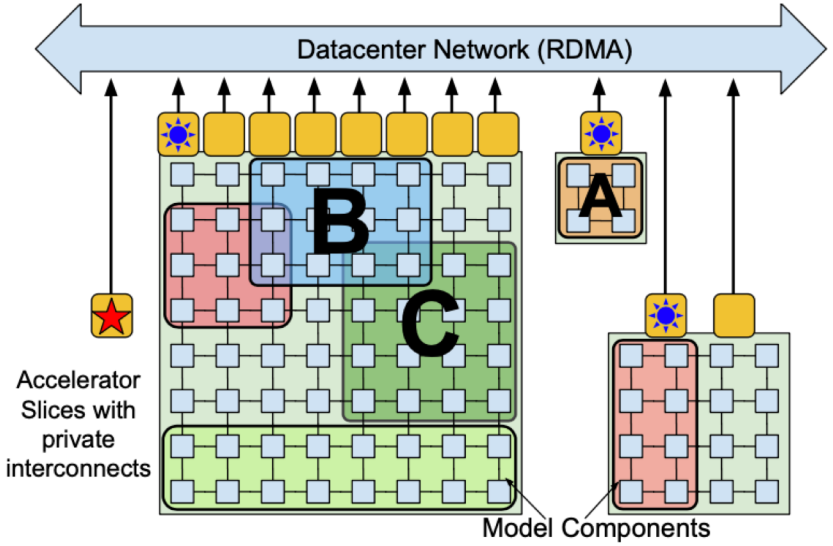


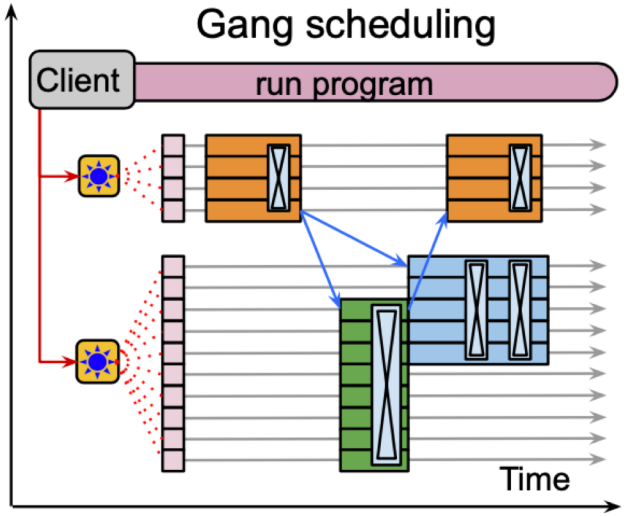
Pathways

System Overview

- Distributed computation expressed as a DAG
- Resource Manager allocates accelerators for each compiled function
- Centralized schedulers gang-schedule computations



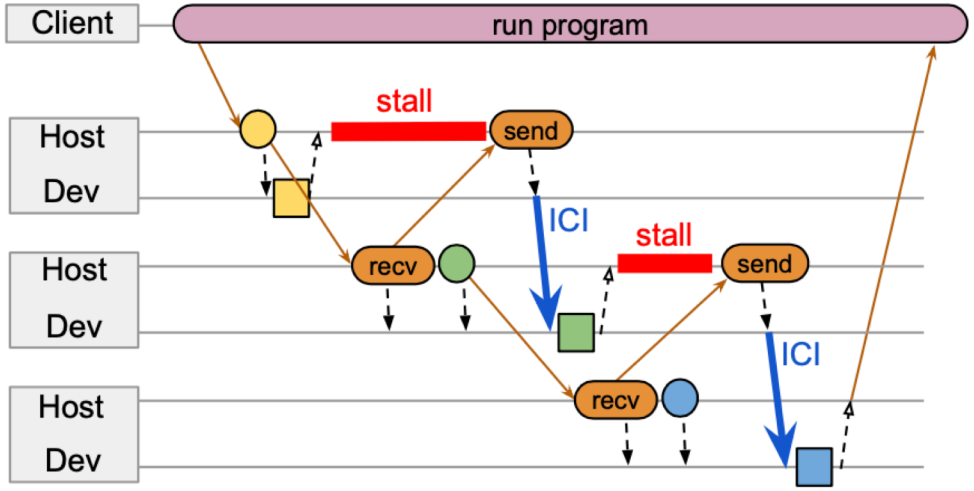




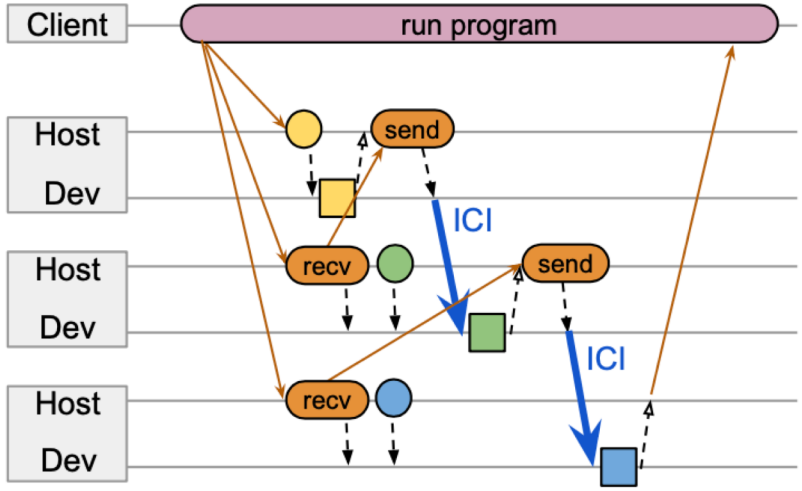
Pathways

Parallel asynchronous dispatch

- time for scheduling, allocation, and coordination is long
- optimization option
- resource requirements are known beforehand



(a) Sequential dispatch



(b) Parallel dispatch

Thanks