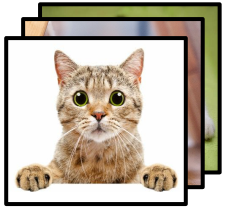
Resource Elasticity in Distributed Deep Learning

Andrew Or, Haoyu Zhang*, Michael J. Freedman

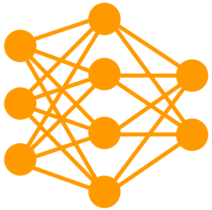
Princeton University, *Google AI

MLSys 2020

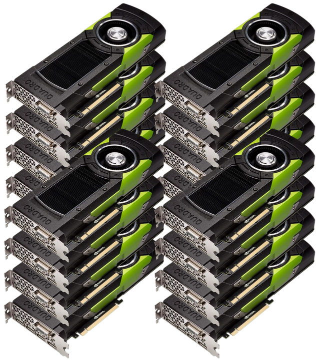
Resource allocation today



Dataset



Model



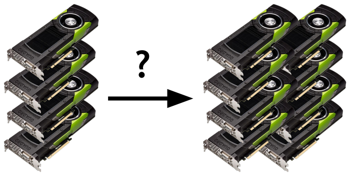
Hardware

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

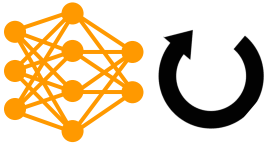
Manual trial-and-error resource allocation

Cumbersome: difficult to estimate scaling behavior



Diverse hardware topologies, communication algorithm etc.

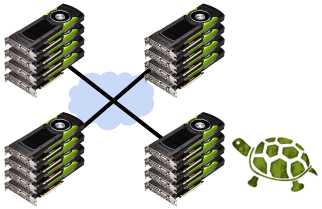
Time-consuming: each trial restarts entire program

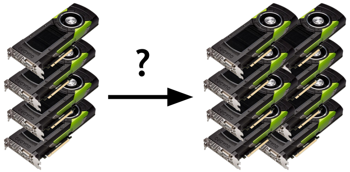


Need to reload libraries, rebuild model, prepare input pipeline etc.

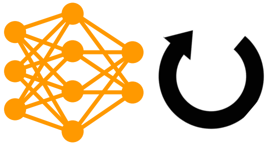
Can take minutes of device idle time

Static allocation: vulnerable to stragglers

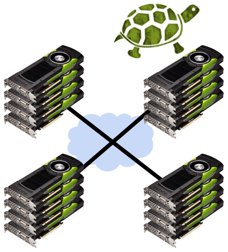




Cumbersome



Time-consuming



Static allocation

Today, users often under- or over-allocate resources

Resource Elasticity in Distributed Deep Learning

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 is not a new idea

Distributed computing	Cloud services	Cluster management	Distributed deep learning
Apache Spark	aws	hadoop YARN	
hadoop MapReduce	Azure	Apache MESOS	
	Google Cloud	kubernetes	

Why is resource elasticity not adopted yet?

Hurdle #1: Lack of applicable scaling heuristics

Hurdle #2: Existing frameworks assume static allocation

Hurdle #3: How to scale the batch size?

Hurdle #1: Lack of applicable scaling heuristics

Existing heuristics are based on dynamic resource demands

- ✗ E.g. request more containers if CPU utilization exceeds X%
- ✗ E.g. kill a worker if it has been idle for X seconds

kubernetes

APACHE Spark

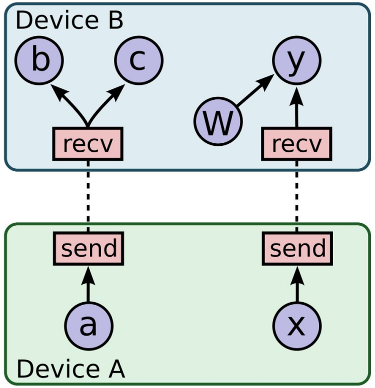
In deep learning workloads, however

Resource utilization is typically consistent across batches, which are short

Workers are rarely idle

Hurdle #2: Existing frameworks assume static allocation

mxnet PyTorch TensorFlow



[Abadi et al., 2015]

Models are structured as static graphs

Communication operations are hard-coded into these graphs

PyTorch has “dynamic” graphs, but dynamic only in inputs

Synchronization primitives assume fixed # devices

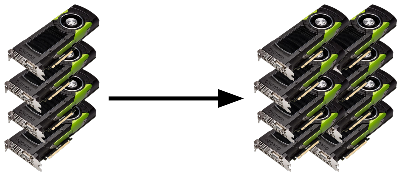
E.g. TensorFlow’s `SyncReplicasOptimizer`, `MultiWorkerMirroredStrategy`

Hurdle #3: How to scale the batch size?

1) Fix per device batch size, vary global batch size

✓ Preserves per device efficiency

✗ Large batch sizes may compromise convergence behavior
[Keskar et al., 2016; Goyal et al., 2017; Hoffer et al., 2017]



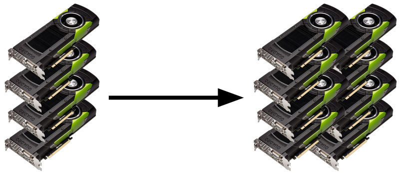
Per device: 32
Global: 128

Per device: 32
Global: 256

2) Fix global batch size, vary per device batch size

✓ Preserves convergence behavior

✗ Sacrifices per device efficiency and overall performance



Per device: 32
Global: 128

Per device: 16
Global: 128

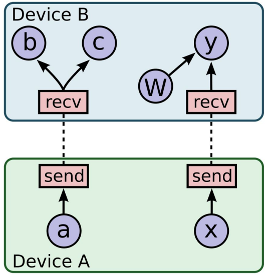
Why is resource elasticity not adopted yet?

APACHE Spark

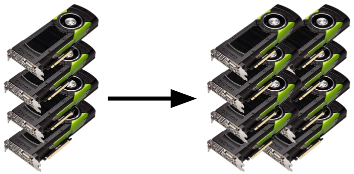
kubernetes

Hurdle #1: Lack of applicable **scaling heuristics**

Hurdle #2: Existing frameworks assume **static allocation**



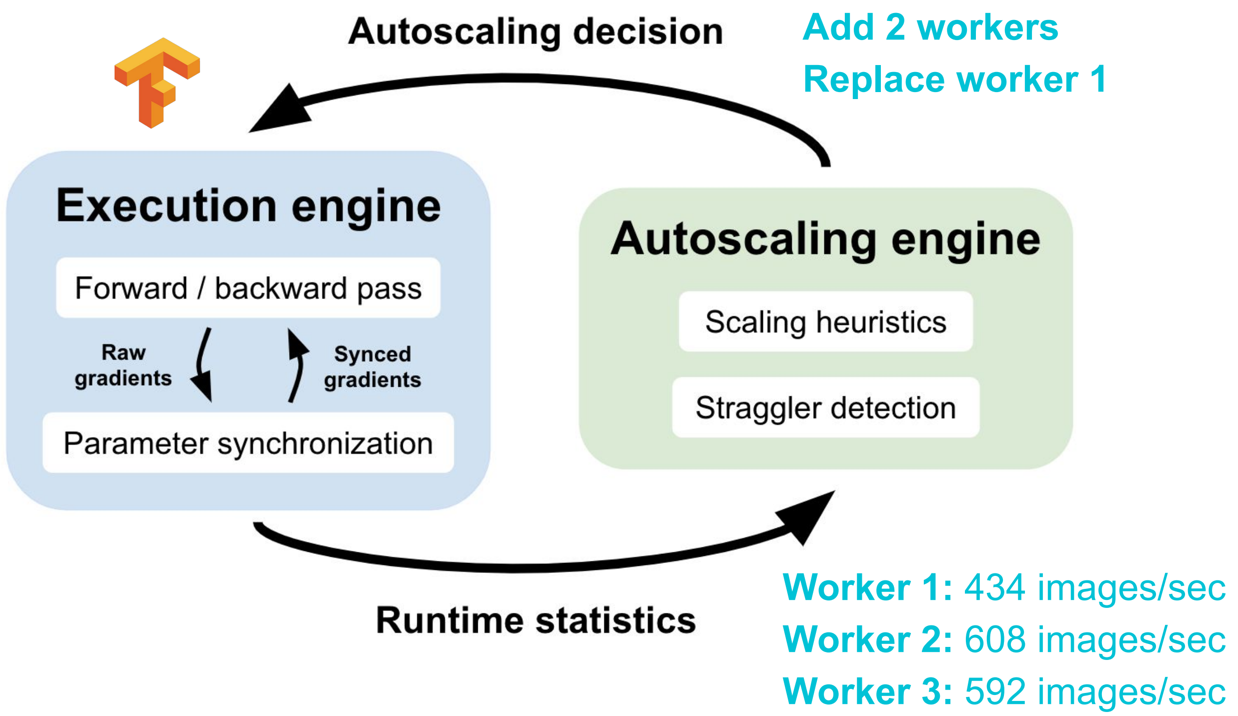
Hurdle #3: How to scale the **batch size**?



Autoscaling System

Scaling heuristics, integration, straggler mitigation

Autoscaling engine for distributed deep learning



Hurdle #1: Lack of applicable scaling heuristics

Design custom scaling heuristics based on:

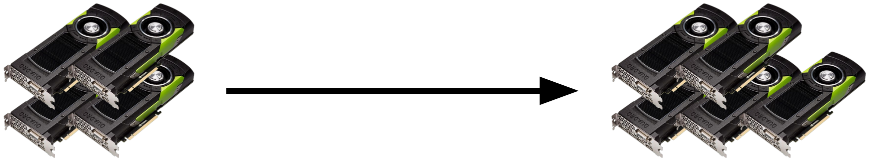
1) **Throughput scaling efficiency**
2) **Utility vs cost**

...

Autoscaling engine can run with custom, pluggable heuristics

Scaling heuristics: Throughput scaling efficiency

Intuition: measure **extra per worker throughput** relative to **existing per worker throughput**



Num workers: 4
Throughput: 400 img/s

Num workers: 5
Throughput: 480 img/s

Throughput scaling efficiency ($s_{k,d}$) = (480 - 400) / (400 / 4) = 0.8

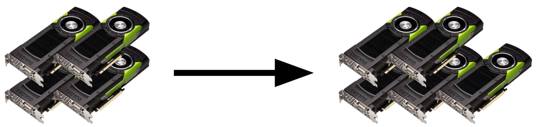
Scaling heuristics: Throughput scaling efficiency

Intuition: measure **extra per worker throughput** relative to **existing per worker throughput**

$s_{k,d} = 1$ perfect scaling

$s_{k,d} = 0$ no improvement

$s_{k,d} < 0$ negative scaling



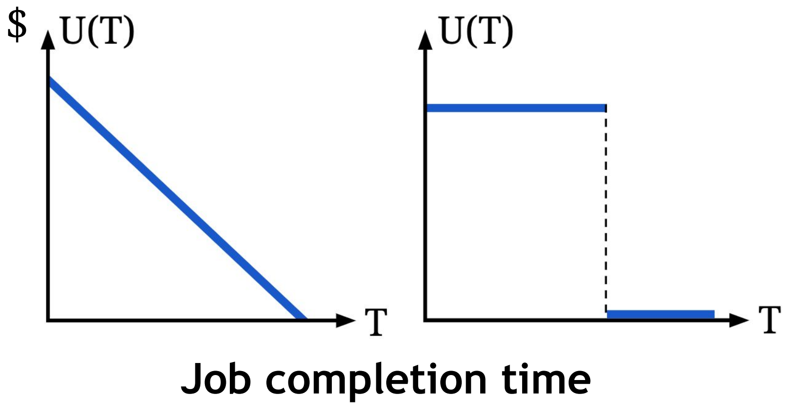
Throughput: 400 img/s → 480 img/s

Efficiency ($s_{k,d}$): (480 - 400) / (400 / 4) = **0.8**

Scaling condition #1: $s_{k,d} > S, S \in [0, 1]$

Scaling heuristics: Utility vs cost

Intuition: compare user-provided utility function to dollar cost of job



Cost $C(k)$ =

Total compute time

×

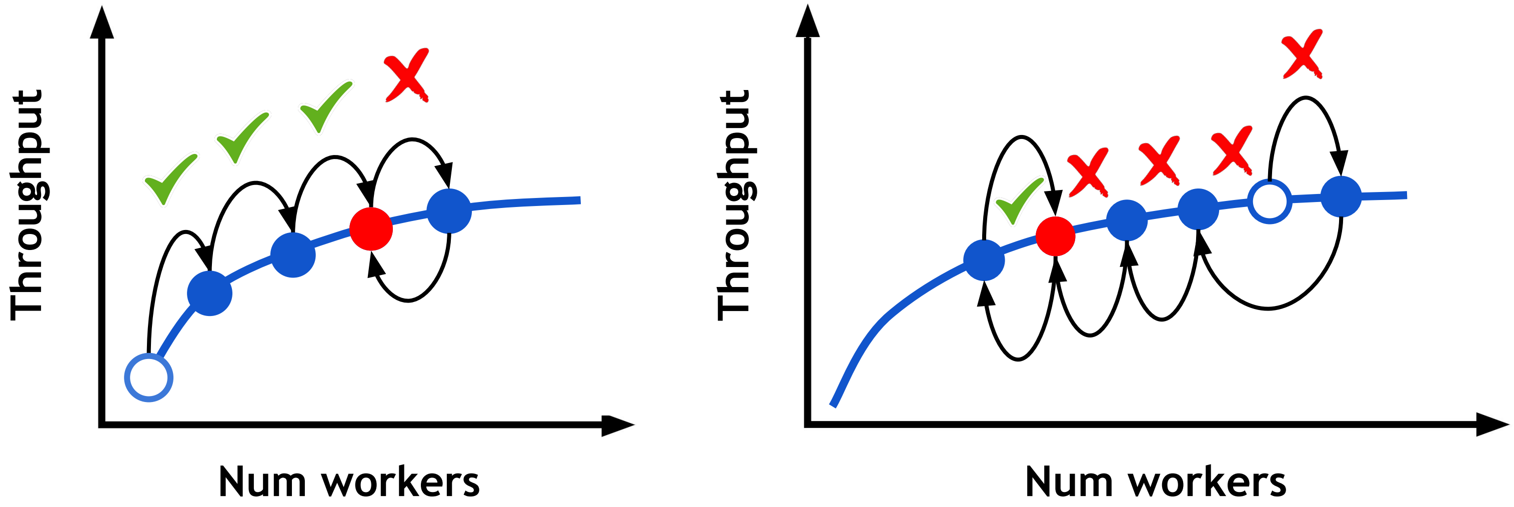
Price per device per time unit

Scaling condition #2: $\Delta U > \Delta C$

Scaling in action

e.g. $s_{k,d} > S$

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



Hurdle #2: Existing frameworks assume static allocation

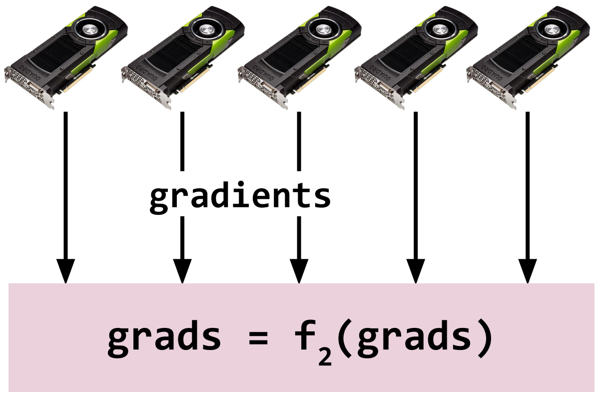
mxnet PyTorch TensorFlow

Give each worker the **illusion of local training**

Workers independently apply **black-box function**
f that synchronizes gradients

Replace function when switching to new allocation

Portable across different frameworks ✓

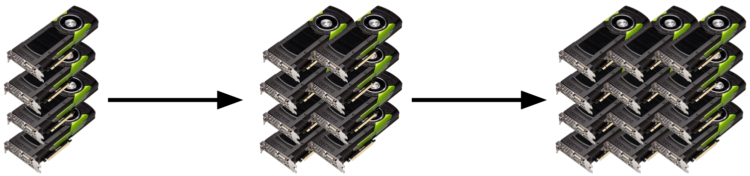


e.g. Horovod allreduce

Hurdle #3: How to scale the batch size?

User provides an **upper batch size limit**

Increase global batch size, fixing per device batch size, until limit



Per device: 32
Global: 128

Per device: 32
Global: 256

Per device: 22
Global: 256

Finding an optimal batch size for arbitrary workloads is an open problem

[Hoffer et al., 2018; Shallue et al., 2018; Smith et al., 2018]

Straggler mitigation comes almost for free

Once we detect a straggler, replace it using the same mechanisms

Refer to paper for details of straggler detection

Evaluation

Job completion time, GPU time, idle time

Experiment setup

CPU cluster: 60 machines

16 Intel Xeon CPUs @ 2.6 GHz *(960 total)*

64GB memory

1 Gbps network

GPU cluster: 8 machines

8 NVIDIA **V100 GPUs** *(64 total)*

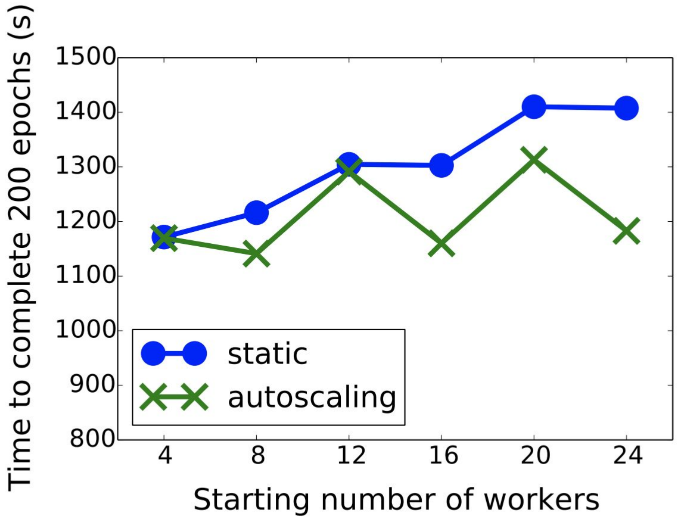
64 Intel Xeon CPUs (2.2GHz)

250GB memory

16 Gbps network

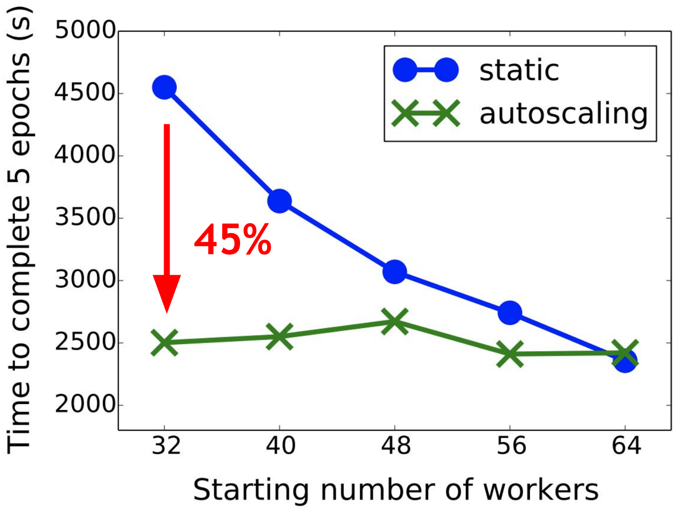
Autoscaling reduces job completion time

ResNet-50 on CIFAR-10



Avg reduction: 8.23%; Max: 16.0%

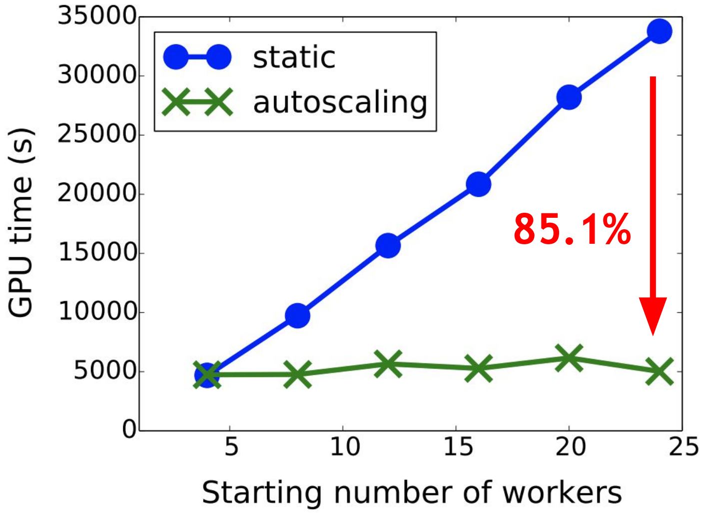
ResNet-50 on ImageNet



Avg reduction: 19.4%; Max: 45.0%

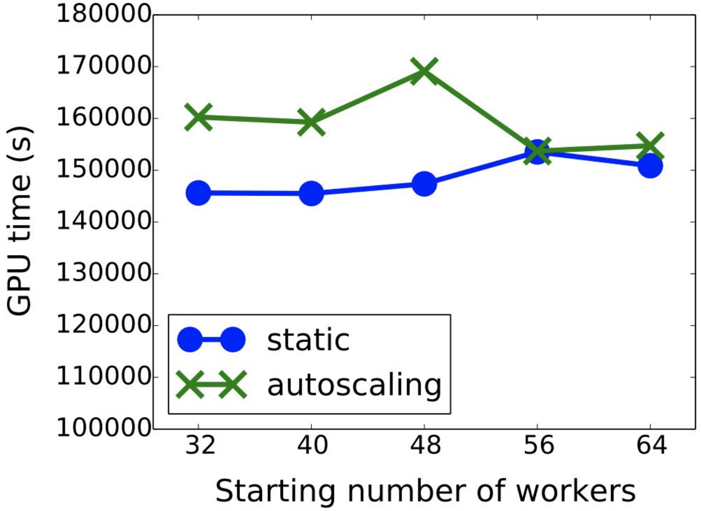
Autoscaling reduces GPU time

ResNet-50 on CIFAR-10



Avg reduction: 58.6%; Max: 85.1%

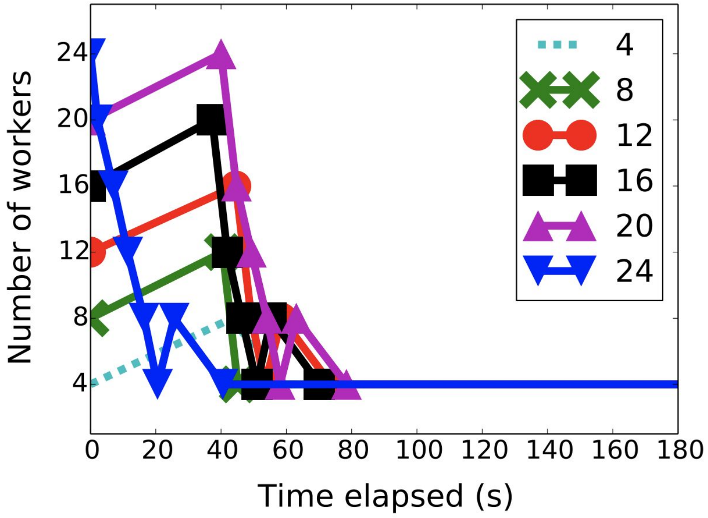
ResNet-50 on ImageNet



Avg *increase*: 7.39%; Max: 14.7%

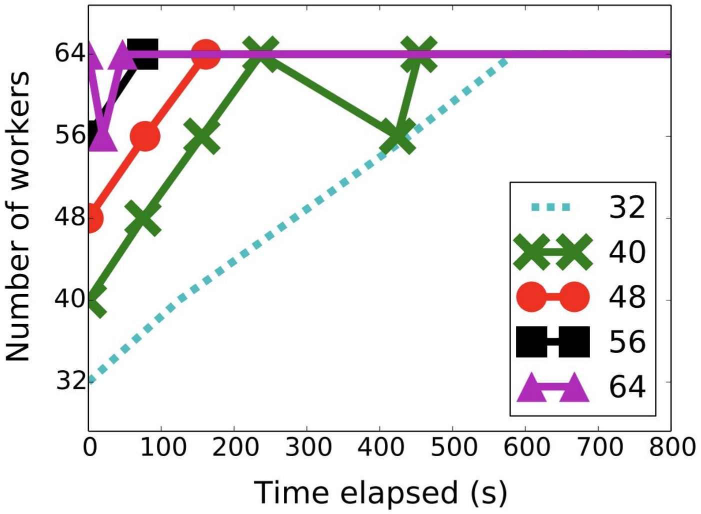
Autoscaling finds target configuration quickly

ResNet-50 on CIFAR-10



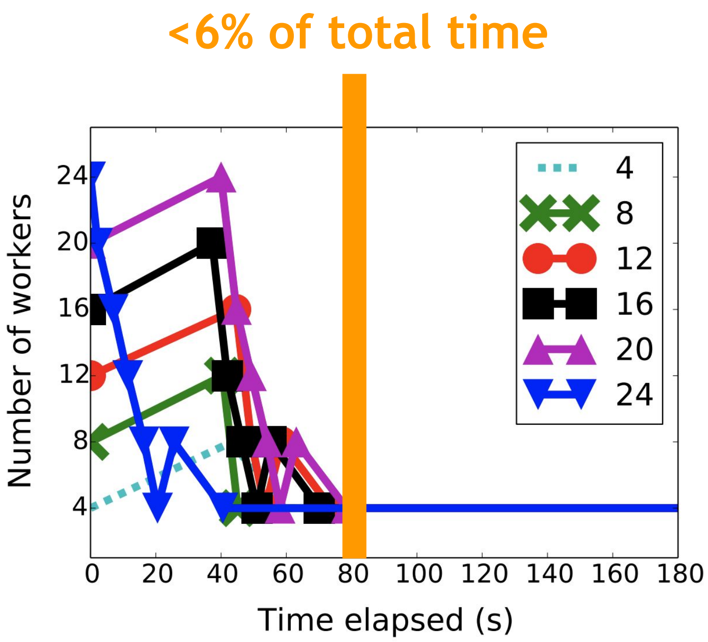
Avg: 61.0s; Max: 78.4s

ResNet-50 on ImageNet

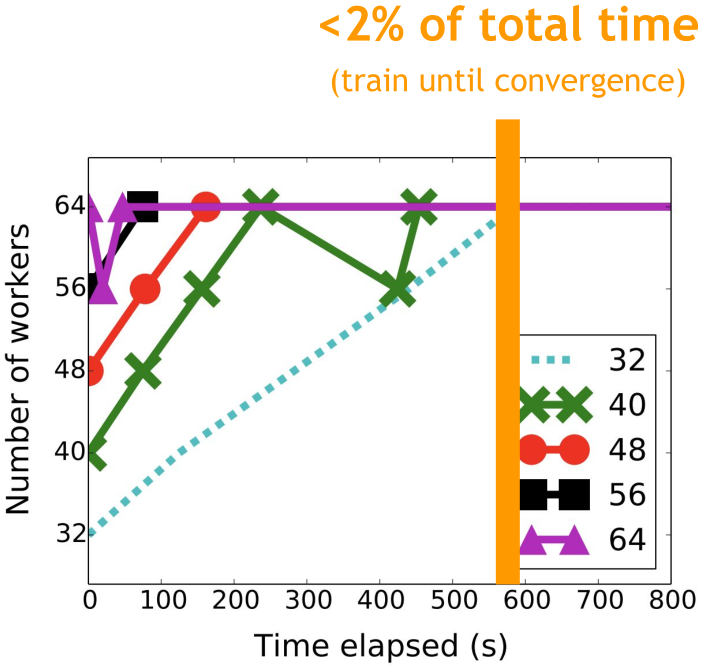


Avg: 264s; Max: 583s

Autoscaling finds target configuration quickly



Avg: 61.0s; Max: 78.4s



Avg: 264s; Max: 583s

Autoscaling has short idle times

	CIFAR-10	ImageNet
autoscaling (+)	3.179	6.813
autoscaling (-)	2.612	4.376
checkpoint restart	72.756	81.186

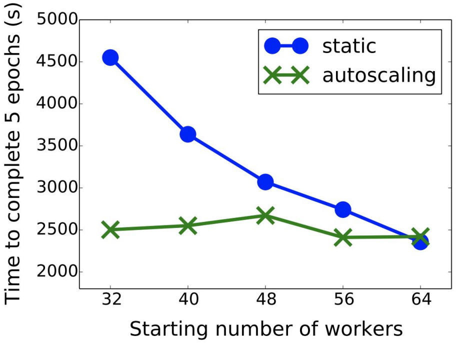
Average idle time during transition (seconds)

Resource Elasticity in Distributed Deep Learning

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