Imperial College London

LSDS
Large-Scale Data & Systems Group

Making Training in Distributed Deep Learning Adaptive

Peter Pietzuch

Imperial College London

http://lsds.doc.ic.ac.uk
<prp@imperial.ac.uk>

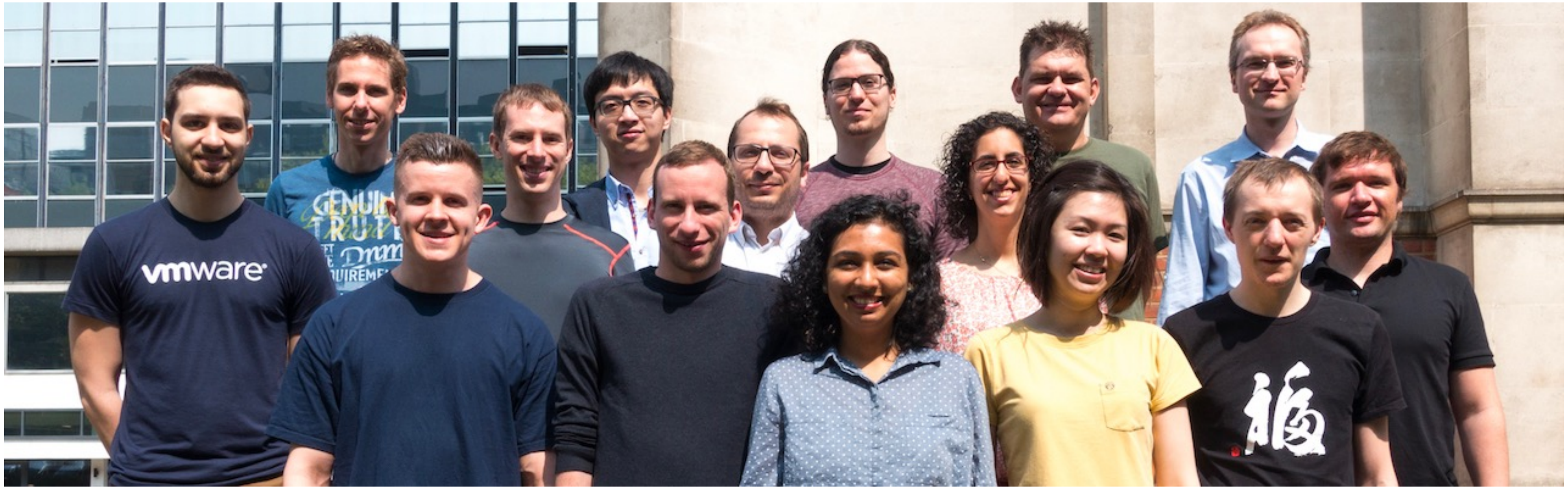
Joint work with Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis and Andrei-Octavian Brabete

Dutch DB Seminar – June 2021

Large-Scale Data & Systems (LSDS) Group

Currently 20 members
(4 faculty, 4 post-docs, 12 PhD students)

http://lsds.doc.ic.ac.uk



LSDS mission statement:

“To support the design and engineering of scalable, robust and secure data-intensive applications”

Research interests and expertise

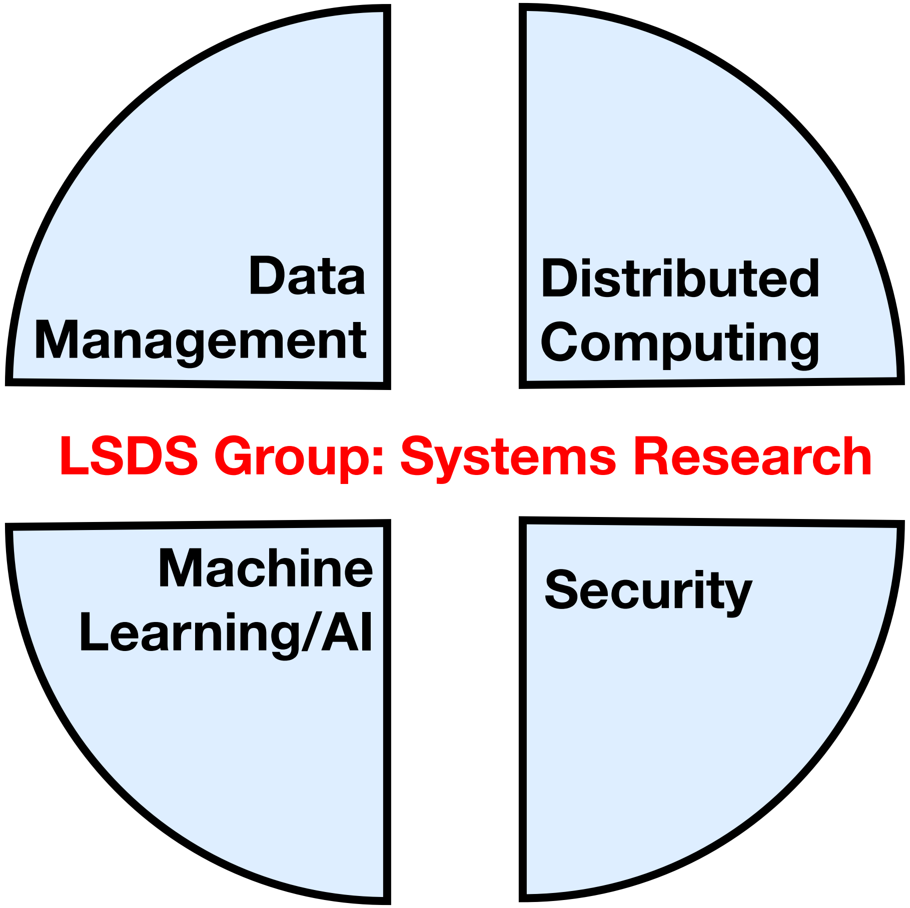
LSDS
Large-Scale Data & Systems Group

- Systems:
  - Distributed systems
  - Operating systems
  - Compilers
  - Networks
  - Runtime systems
- Hardware & Infrastructure:
  - Multicore CPUs
  - Trusted Hardware, TEEs
  - Accelerators/GPUs
  - Data-center networks, RDMA
  - Edge infrastructure
- Application domains:
  - Data management
  - Stream processing
  - Graph processing
  - Machine learning/AI
  - Blockchain
- Techniques:
  - Resource management
  - Scheduling
  - Query optimisation
  - Network programmability

Past & Present LSDS Research

- **Distributed dataflow systems**
  [SIGMOD'18, ICDE'16, ATC'14, SIGMOD'13]
- **Multicore data processing**
  [SIGMOD'16, VLDB'14]
- **Heterogeneous architectures**
  [CIDR'19, SIGMOD'16]
- **Stream processing**
  [SIGMOD'20, EDBT'20, VLDB'17, SIGMOD'16, CIDR'15, ICDE'11]
- **IoT data processing**
  [VLDB'18]

- **Serverless computing**
  [USENIX ATC'20]
- **Container scheduling**
  [SoCC'19, EuroSys'18]
- **Edge computing**
  [TMC, MobiSys'15]
- **In-network processing**
  [USENIX ATC'17, USENIX ATC'16, CoNEXT'14]



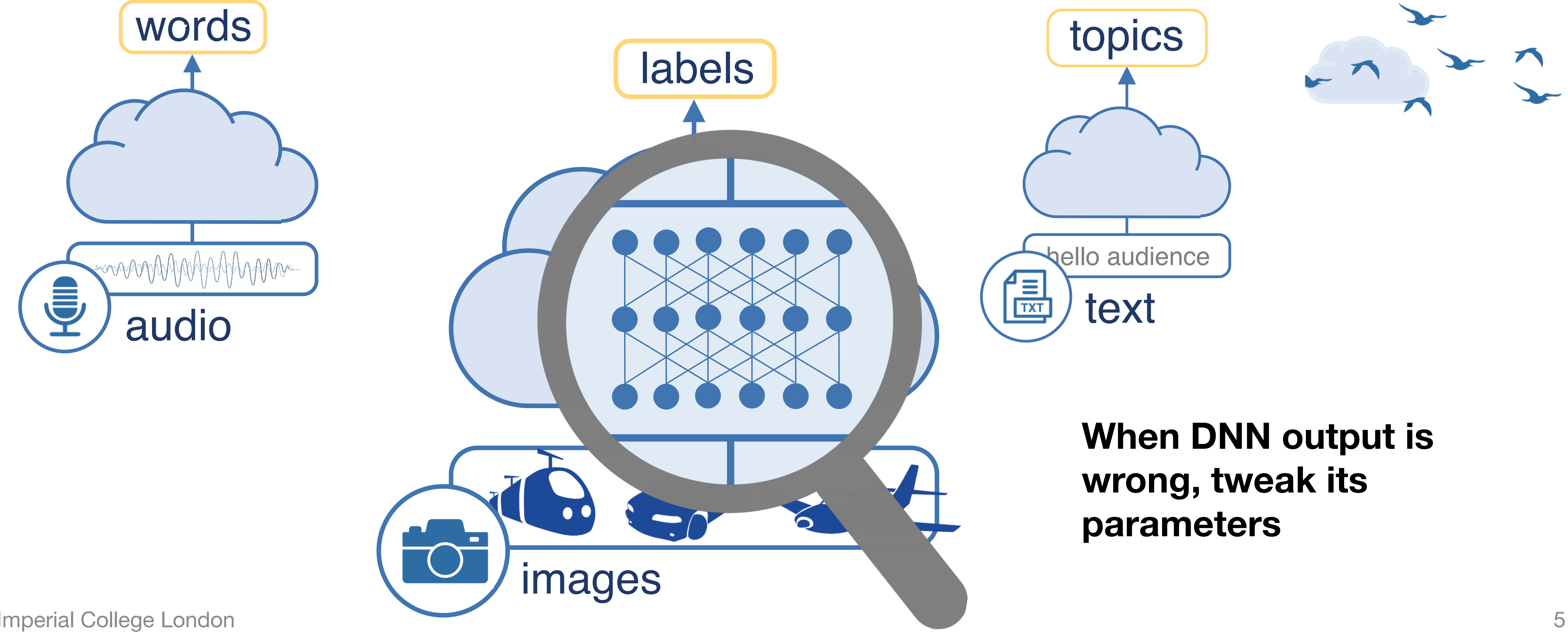
- **Scalable machine learning**
  [OSDI'20, HotCloud'20, VLDB'19, SysML'18]
- **Expressive machine learning**
  [OSDI'20, HotCloud'20, SysML'18]
- **Decentralised machine learning**
  [SoCC'16]

- **Trusted hardware**
  [ASPLOS'21, EuroSys'21, VEE'21, EuroSys'18, USENIX ATC'17, OSDI'16]
- **Blockchain**
  [SOSP'17, BITCOIN'17]
- **Information flow control**
  [Middleware'16, ICDE'14, ATC'10]
- **Cloud/web security**
  [CCS'15, WebApps'11]

Deep Neural Networks (DNNs) Have a Big Impact

Revolutionised solutions in **vision**, **speech recognition**, …

DNN models are **trained** by giving **examples** (instead of programming)



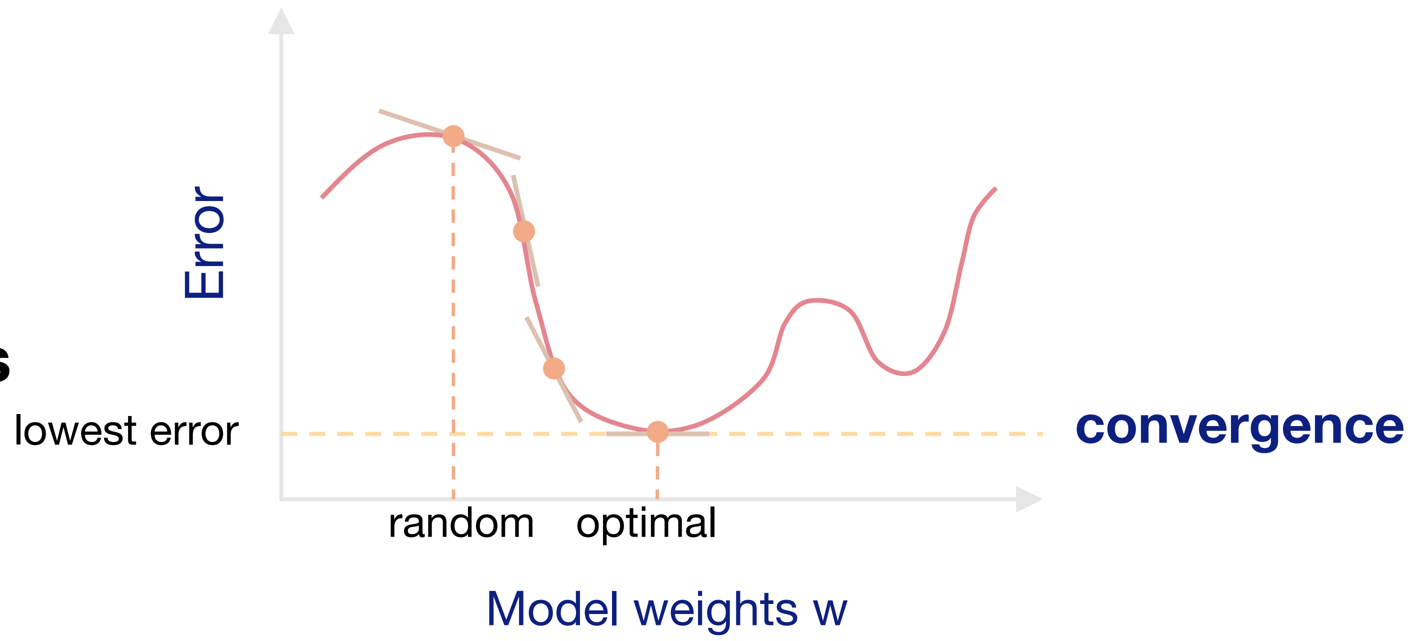
When DNN output is wrong, tweak its parameters

Training Deep Neural Networks (DNNs)

Obtain DNN model that minimises **classification error**

Use **Stochastic Gradient Descent (SGD)** for training:

1. Begin with **random model**
2. Consider **mini-batch** of training data
3. Iteratively calculate **gradients** & update **model weights w**



Deep Learning on GPUs

GPUs are good at parallelising gradient computation

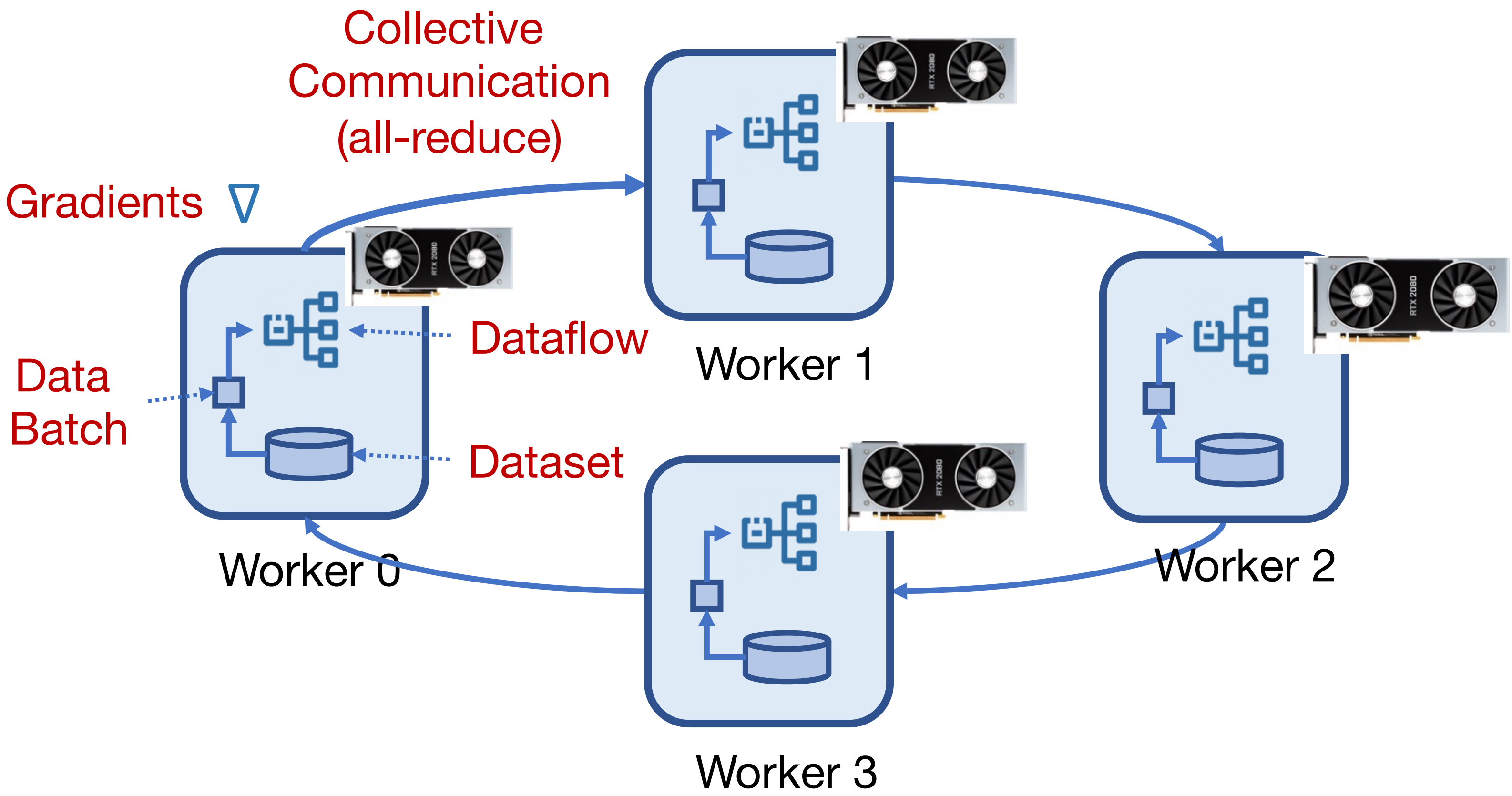
Distributed Deep Learning Systems

Combine large **training data** and **models**

TensorFlow

PyTorch

HOROVOD

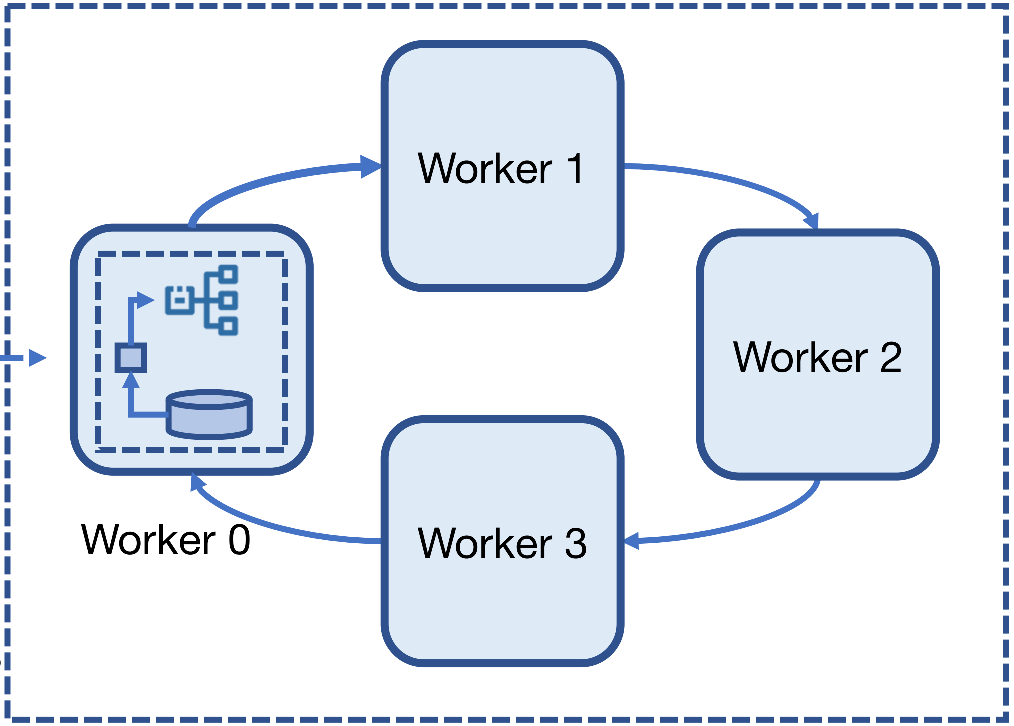


Parameters in Distributed Deep Learning Systems

Users must tune parameters to optimise **time-to-accuracy**

Hyper-parameters
- Batch size
- Learning rate
- …

Small batch or large batch size?



System parameters
- Number of workers
- Communication topology
- …

Ring or binary-tree?

Issues with Parameter Tuning

Examples of empirical parameter tuning	**Issue**
"Change batch size at epoch 30, 60 and 90 when training with ImageNet." [1]	Dataset-specific
"Linearly scale the learning rate with the #workers when training ResNet models." [2]	Model-specific
"Set the topology to a ring by default." [3]	Cluster-specific

[1] Dynamic Mini-batch SGD for Elastic Distributed Training: Learning in the Limbo of Resources, 2020
[2] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018
[3] Horovod: Fast and easy distributed deep learning in TensorFlow, 2018

Dynamic Parameter Adaptation

Example OpenAI predicts batch size based on **Gradient Noise Scale (GNS)**

↓

Intuition GNS measures **variation in data batches**

↓

Proposal

- When GNS is small → keep **batch size**
- When GNS is large → increase **batch size**

Proposals for Dynamic Parameter Adaptation

AdaScale SGD: A User-Friendly Algorithm for Distributed Training

Tyler B. Johnson [† 1] Pulkit Agrawal [† 1] Haijie Gu [1] Carlos Guestrin [1]

Gradient variance

Large-Scale Distributed Second-Order Optimization Using Kronecker-Factored Approximate Curvature for Deep Convolutional Neural Networks

Kazuki Osawa[1] Yohei Tsuji[1,5] Yuichiro Ueno[1] Akira Naruse[3] Rio Yokota[2,5] Satoshi Matsuoka[4,1]

[1]School of Computing, Tokyo Institute of Technology
[2]Global Scientific Information and Computing Center, Tokyo Institute of Technology
[3]NVIDIA
[4]RIKEN Center for Computational Science

Gradient second-order metrics

RESOURCE ELASTICITY IN DISTRIBUTED DEEP LEARNING

Andrew Or [1] Haoyu Zhang [1 2] Michael J. Freedman [1]

Worker performance metrics

Another Example of Adaptation

Distributed deep learning is **resource-intensive**

Accelerated hardware resources (e.g. GPUs) are **expensive**

Example: Training Megatron-LM[3]

- Training of BERT-like model
- 512 NVIDIA V100 GPUs
- One epoch (68,507 iterations) takes 2.1 days

Cost on Azure: $92,613

[3] Shoeybi, Mohammad, et al. Megatron-LM: Training Multi-billion Parameter Language Models using GPU Model Parallelism, 2017

Using Transient Cloud Resources for Training

E.g. AWS Spot instances, Azure Spot VMs

Follow laws of free market

Revocations with short notification

Economic incentive: cost reduction of up to 90%[1]

A Megatron-LM epoch would drop from $92,613 to $15,152

[1] https://azure.microsoft.com/en-us/pricing/spot/

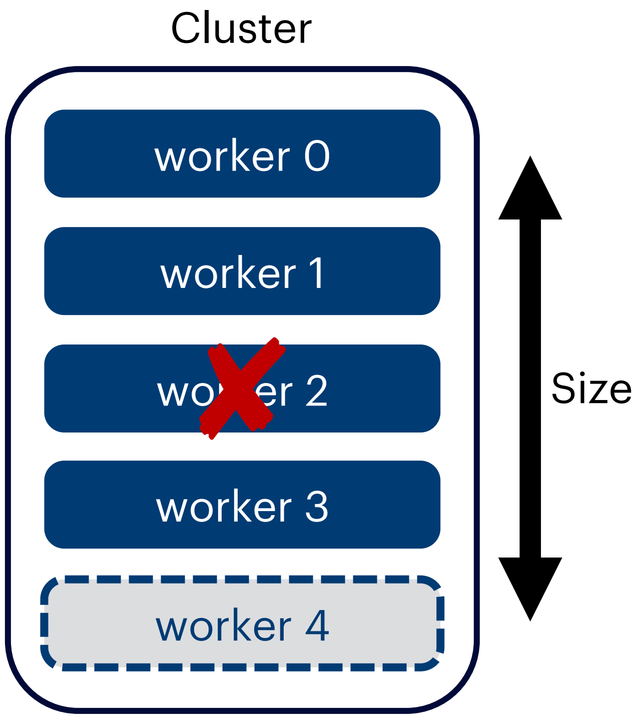
Transient Resources Require Adaptation

New workers become available or old workers get revoked

→ System must cope with disappearing resources

Changes can happen at any time

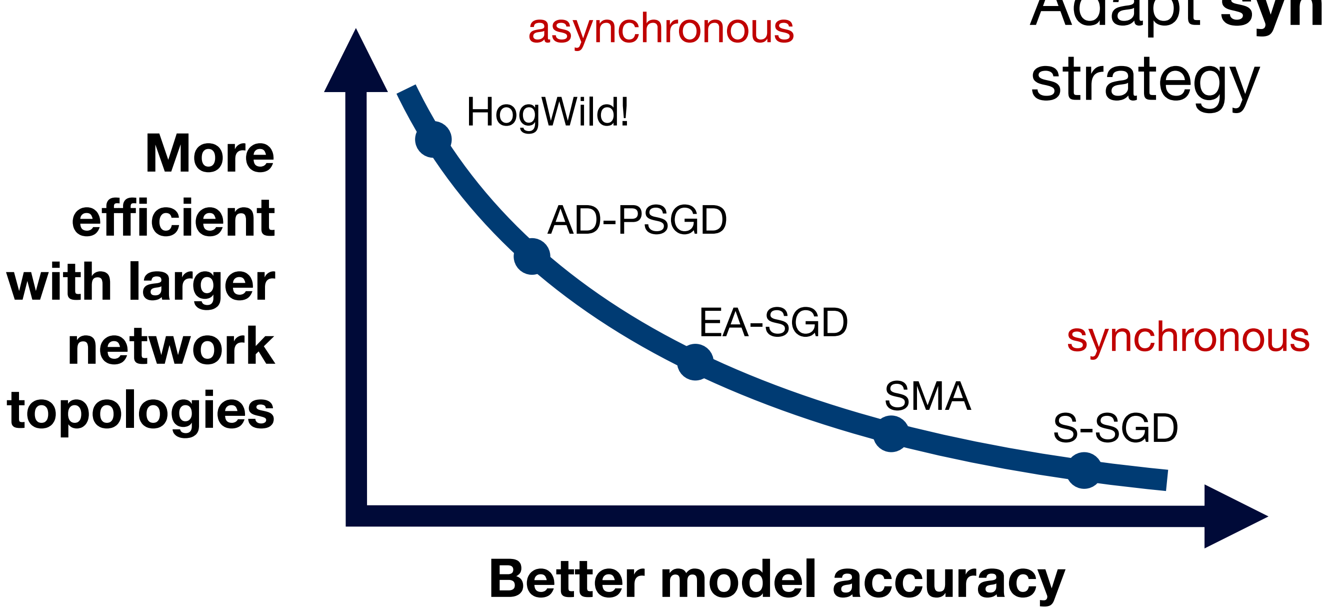
→ System must ensure consistency of updates



Elastic Scaling Requires Adaptation

Cluster size/number of GPUs changes over time

→ System must adapt to different network topologies



Adapt **synchronisation** strategy

Open Challenges

Can we design distributed deep learning systems that supports adaptation?

Design challenges:

- **How to support different types of adaptation?**
- **How to adapt based on large data volumes?**
- **How to change parameters of workers consistently?**

Existing Approaches for Adaptation

1. Specific mechanisms for adaptation

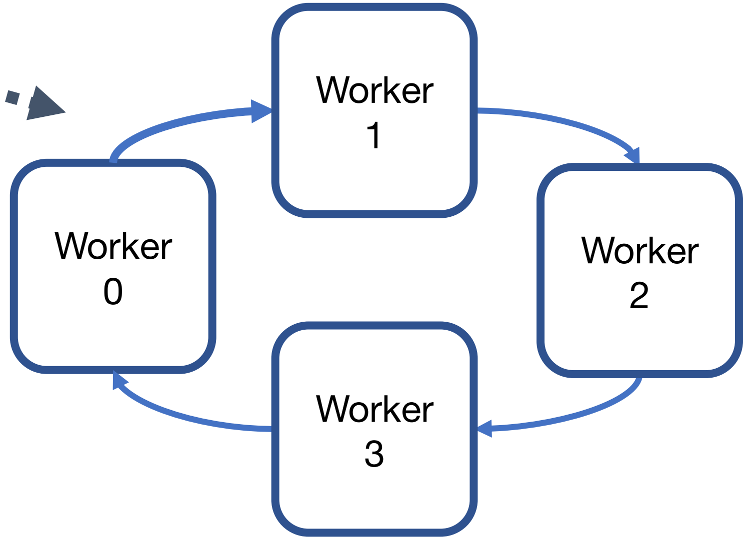
AutoScaling [MLSys'20]

Horovod Elastic

PyTorch Elastic

TensorFlow Elastic

Custom adaptation without generic APIs



2. Processing of monitoring data offline

Logs

TensorBoard

mlflow

Expensive data movement

3. Checkpoint-and-recover

Write checkpoint → Release resources → Acquire resources → Start training process → Read checkpoint

Not possible to change parameters during runtime

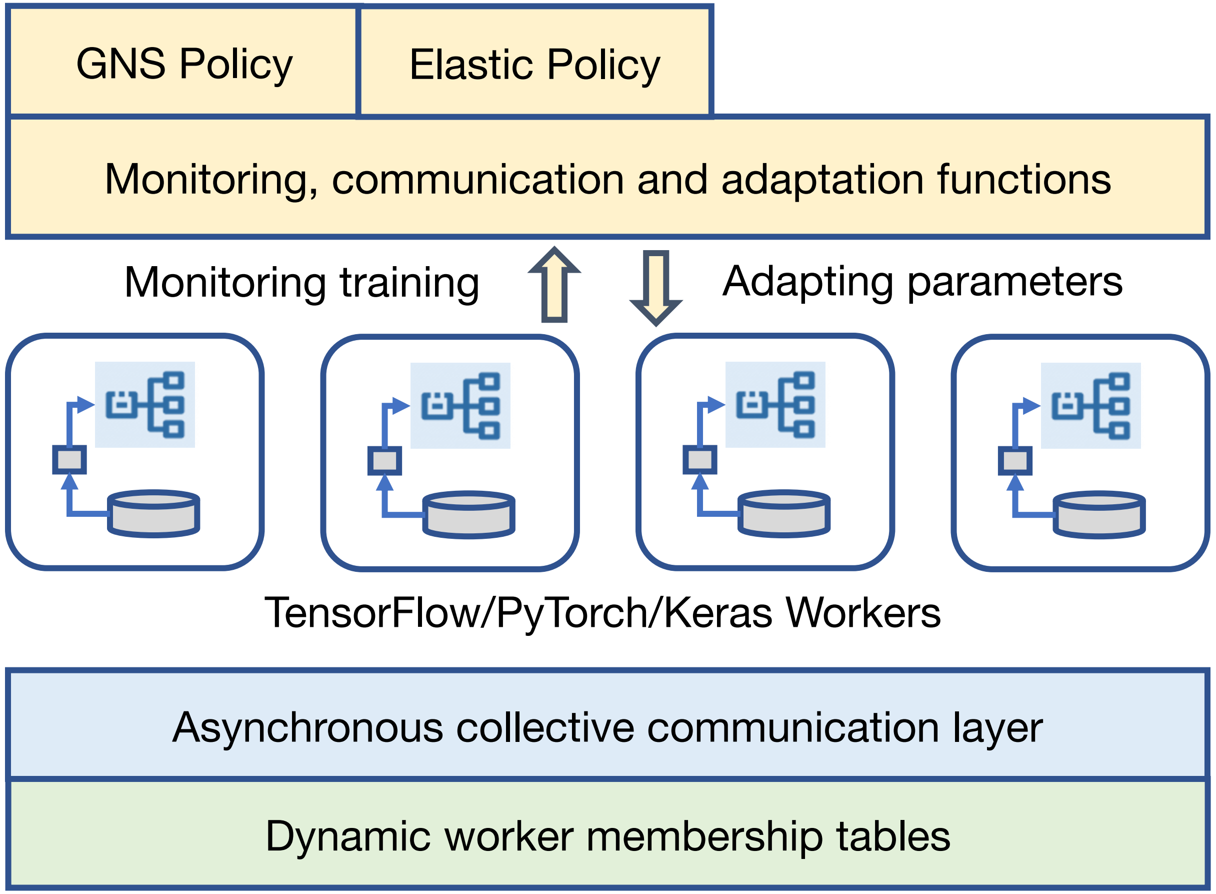
KungFu – Distributed Training Library

Contributions:

1. Supporting adaptation policies

2. Monitoring inside dataflow

3. Distributing parameter updates



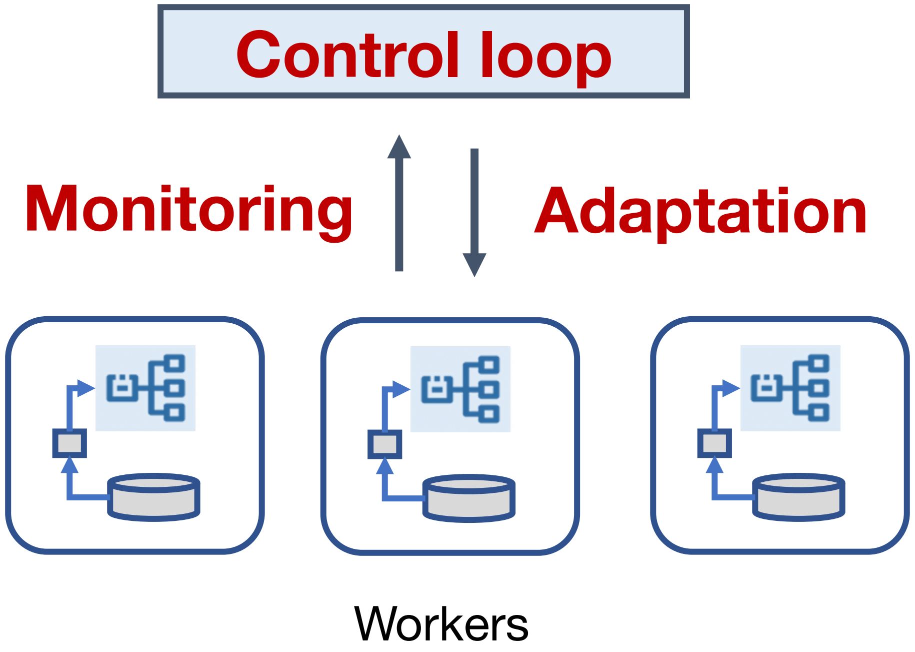
Supports **different types of adaptation**

Processes **large volume of monitoring data**

Adapts **stateful workers consistently**

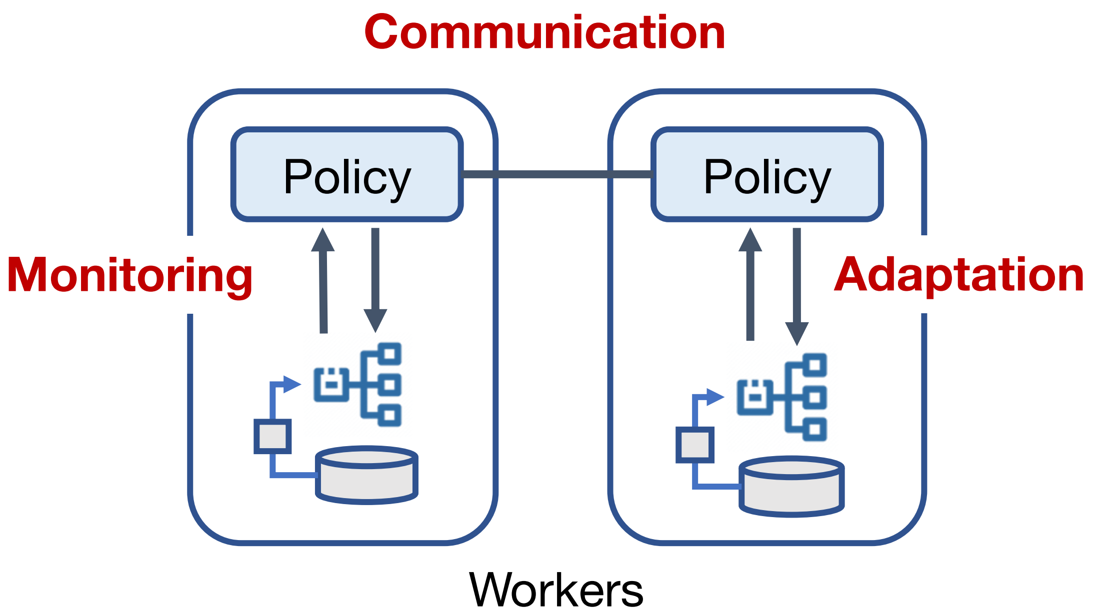
1. Supporting Adaptation Policies

Express Adaptation as Control Loops



Control loop monitors workers and uses monitored metrics to change parameters

Adaptation Policies



Write adaptation policies using
expressive API functions:

Monitoring	Communication	Adaptation
• `grad_noise_scale` • `grad_variance` • …	• `allreduce` • `broadcast` • …	• `resize` • `set_tree` • …

Example: Adaptation Policy for GNS

TensorFlow

PyTorch

1. Adaptation logic in policy

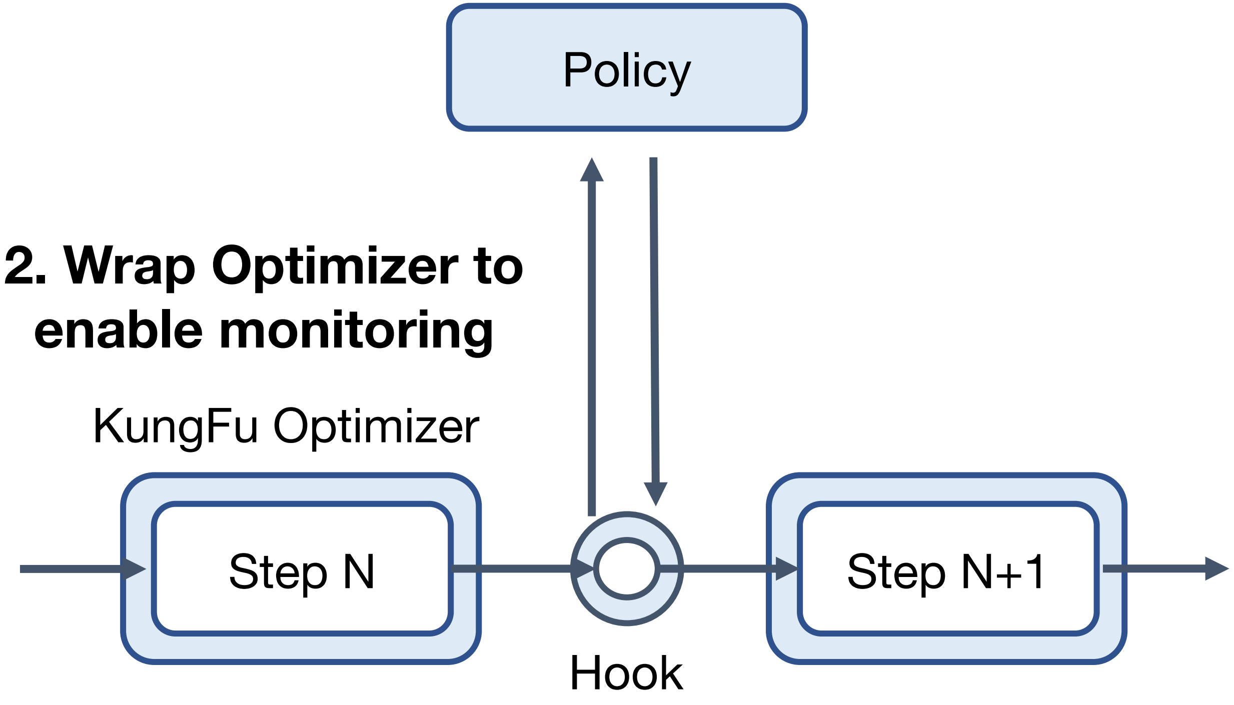
```
import kungfu as kf

class GNSPolicy(kf.BasePolicy)
  def after_step(self):
    gns = kf.grad_noise_scale()
    avg = kf.allreduce(gns, `avg`)
    if avg > self.prev:
      kf.resize(kf.size() + 1)
```

```
opt = SGDOptimizer(…)
opt = kf.Optimizer(opt)
```

```
hook = kf.Hook(GNSPolicy(…))
model, data = …
model.train(data, opt, hook)
```

2. Wrap Optimizer to enable monitoring



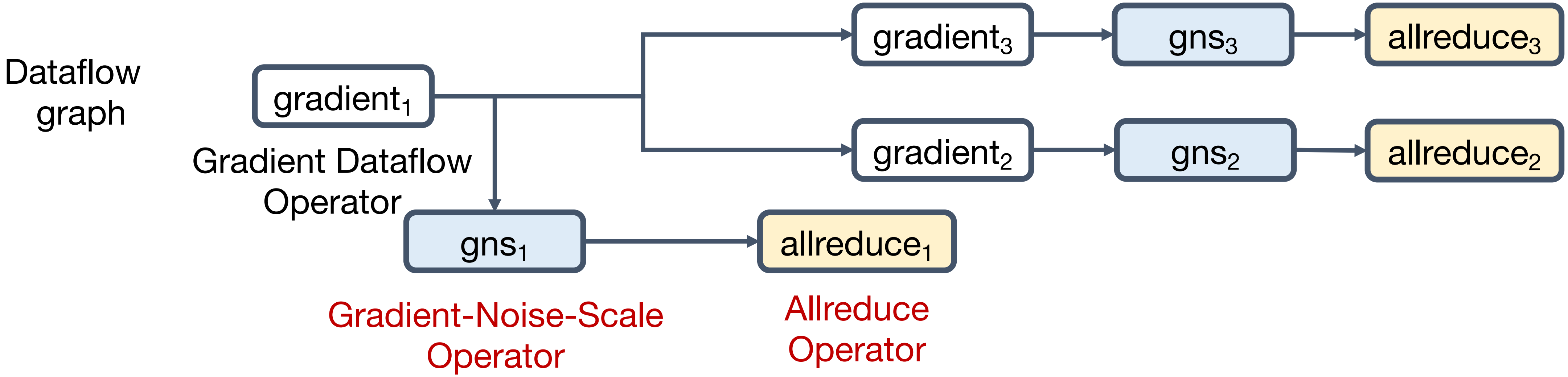
3. KungFu Hooks add policy

2. Monitoring Inside Dataflow

Efficient Monitoring During Training

Problem: High monitoring cost reduces adaptation benefit

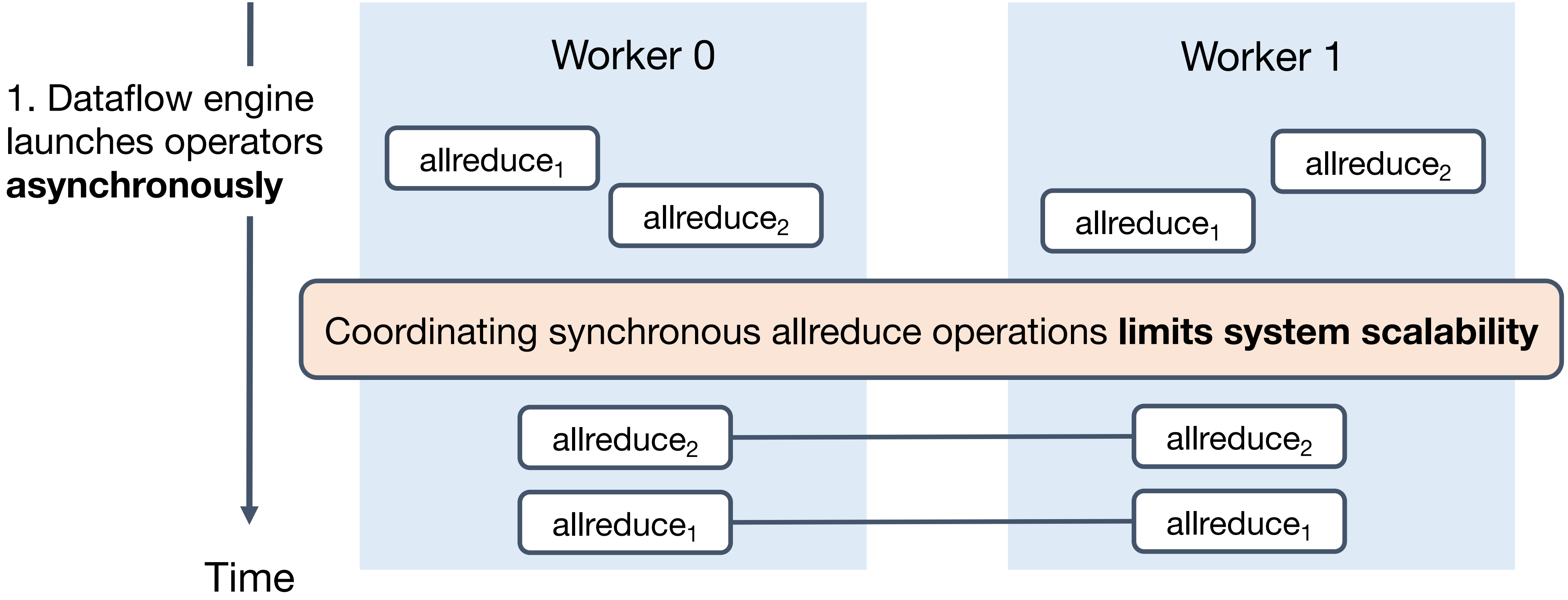
Idea: Include monitoring operators inside dataflow



Monitoring takes advantage of **optimisations** in dataflow engines and **collective communication** support

Efficient Collective Communication

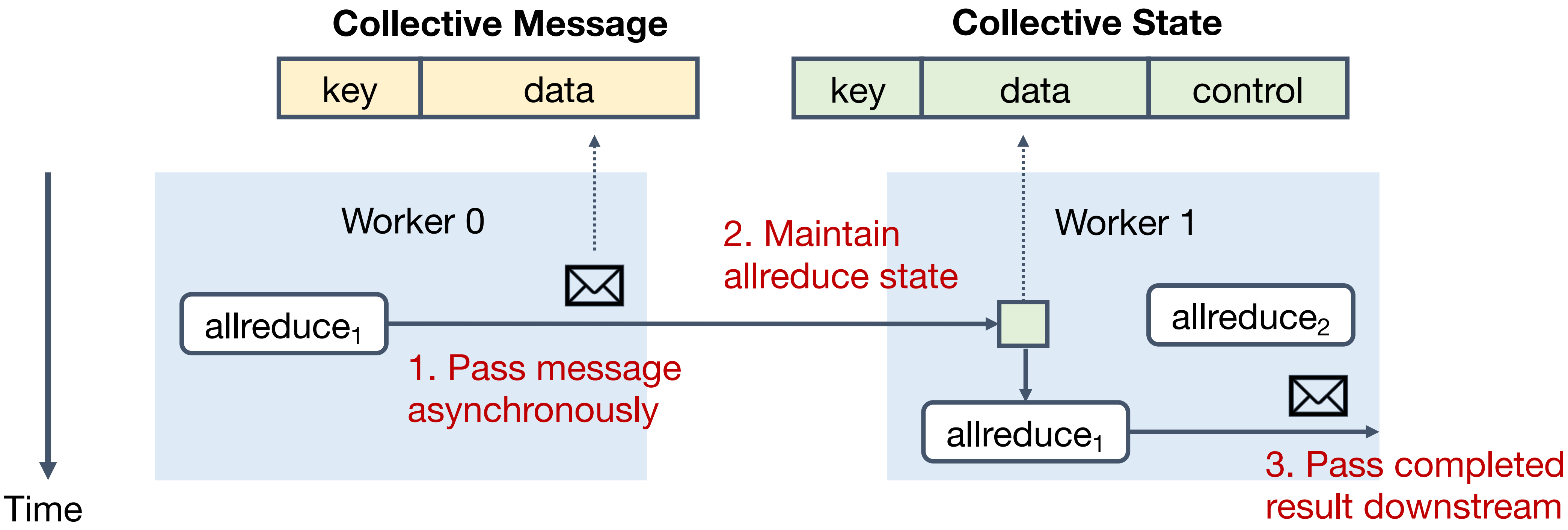
Problem: Extensive use of collective communication reduces performance



2. Message-Passing-Interface (MPI) implementation assumes **synchronous execution**

Asynchronous Collective Communication

Idea: Make collective communication asynchronous

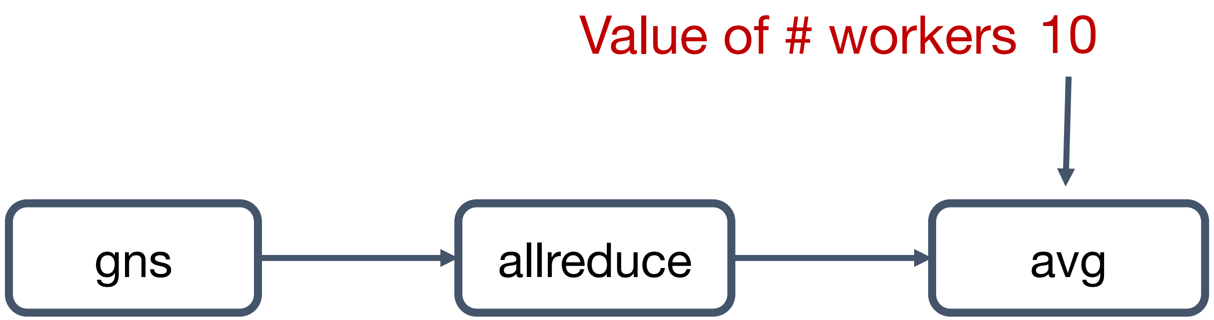


No need for coordination in asynchronous collective communication

3. Distributing Parameter Updates

Changing System Parameters

Problem: Parameter adaptation affects **state consistency**



Dataflow for averaging GNS

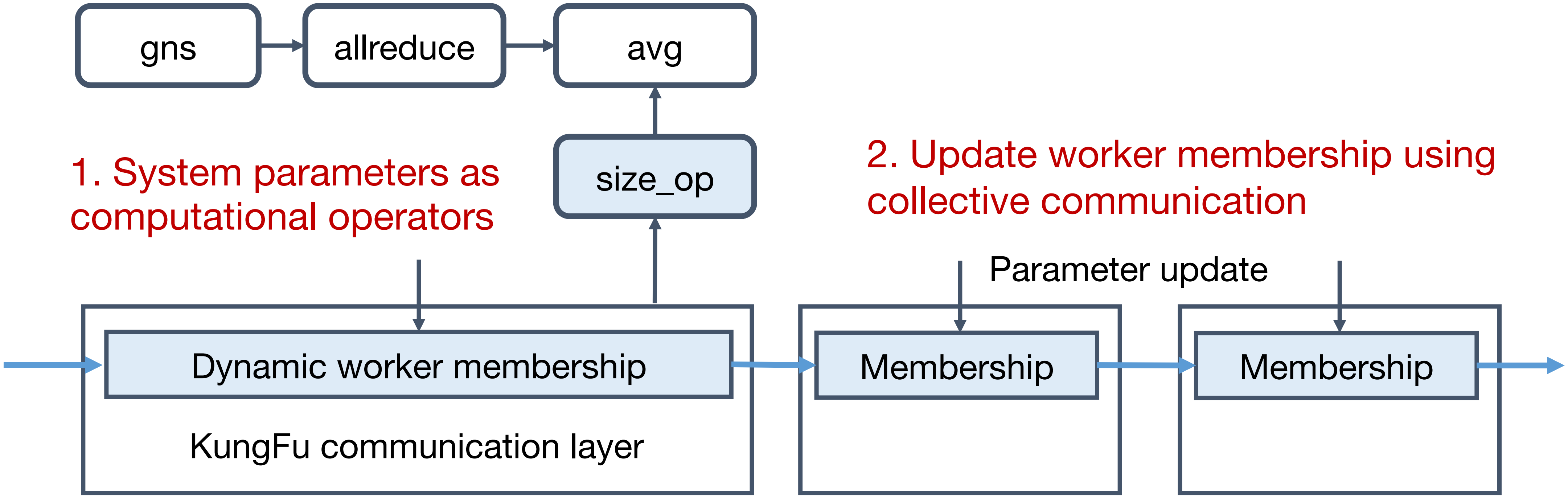
Value may be stale

Other system parameters:
- Worker rank
- Communication topology
- …

Changing system parameters therefore typically requires **system restart**

Distributed Mechanism for Changing Parameters

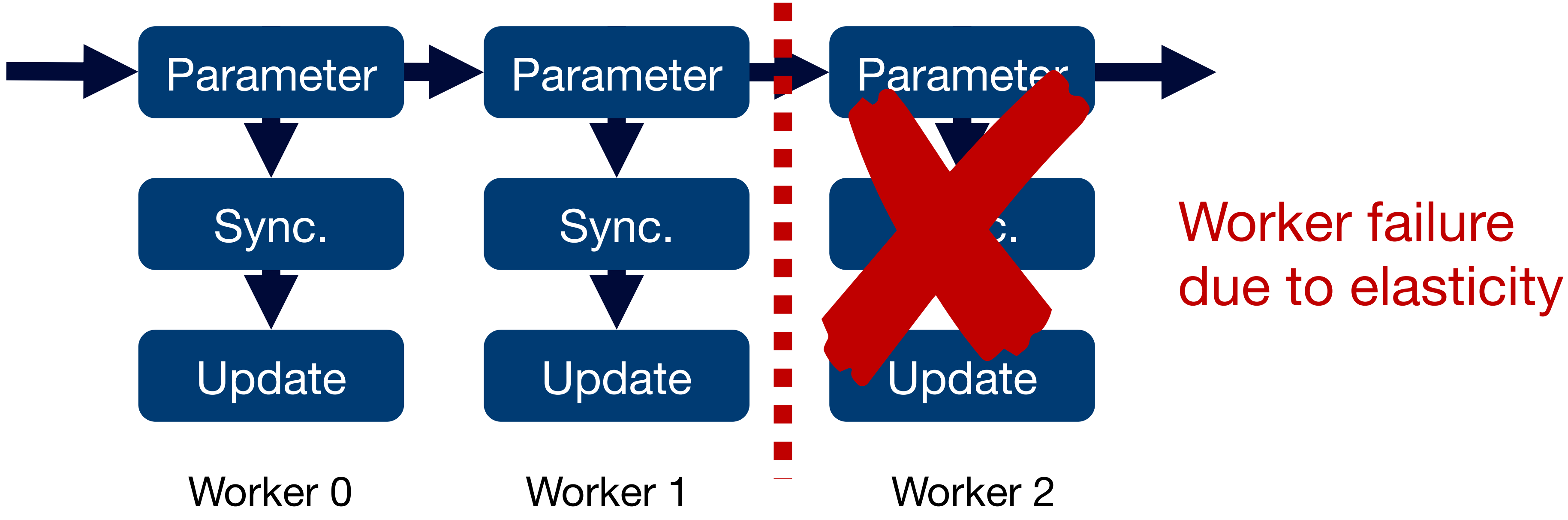
Idea: Decouple system parameters from dataflow state



Always obtains up-to-date view of system parameters

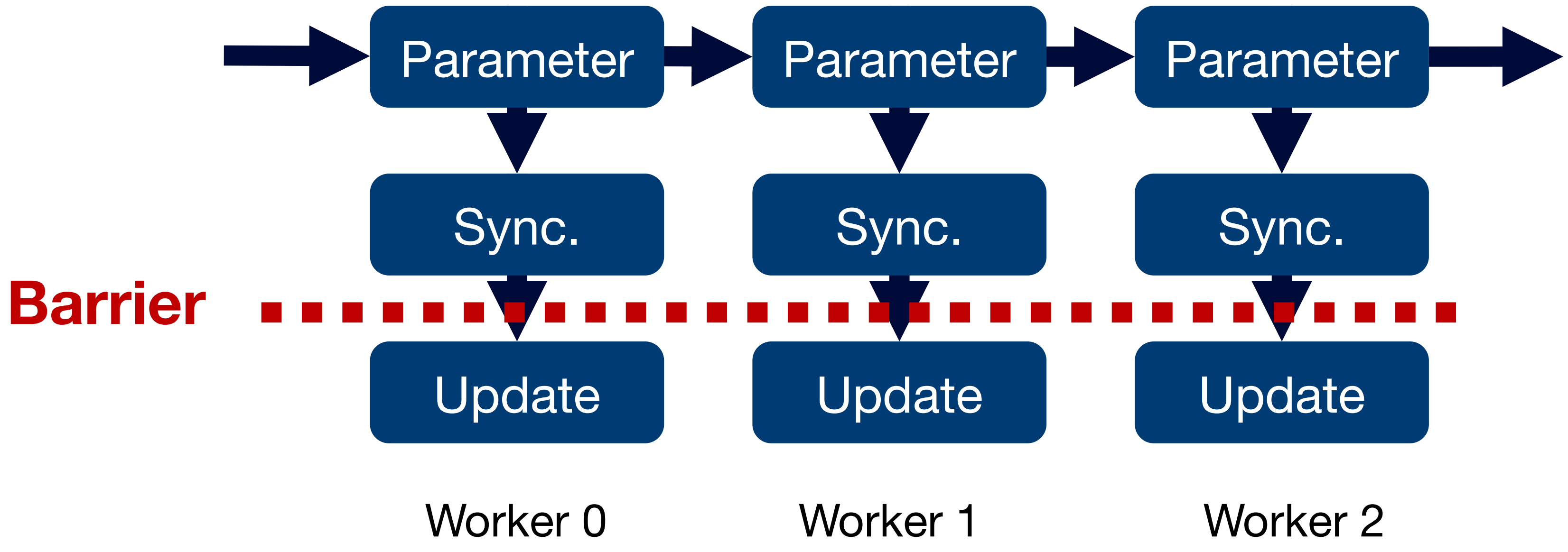
Inconsistent Parameter Updates

Problem: Incomplete parameter changes may lead to **inconsistency**



Atomic Parameter Updates

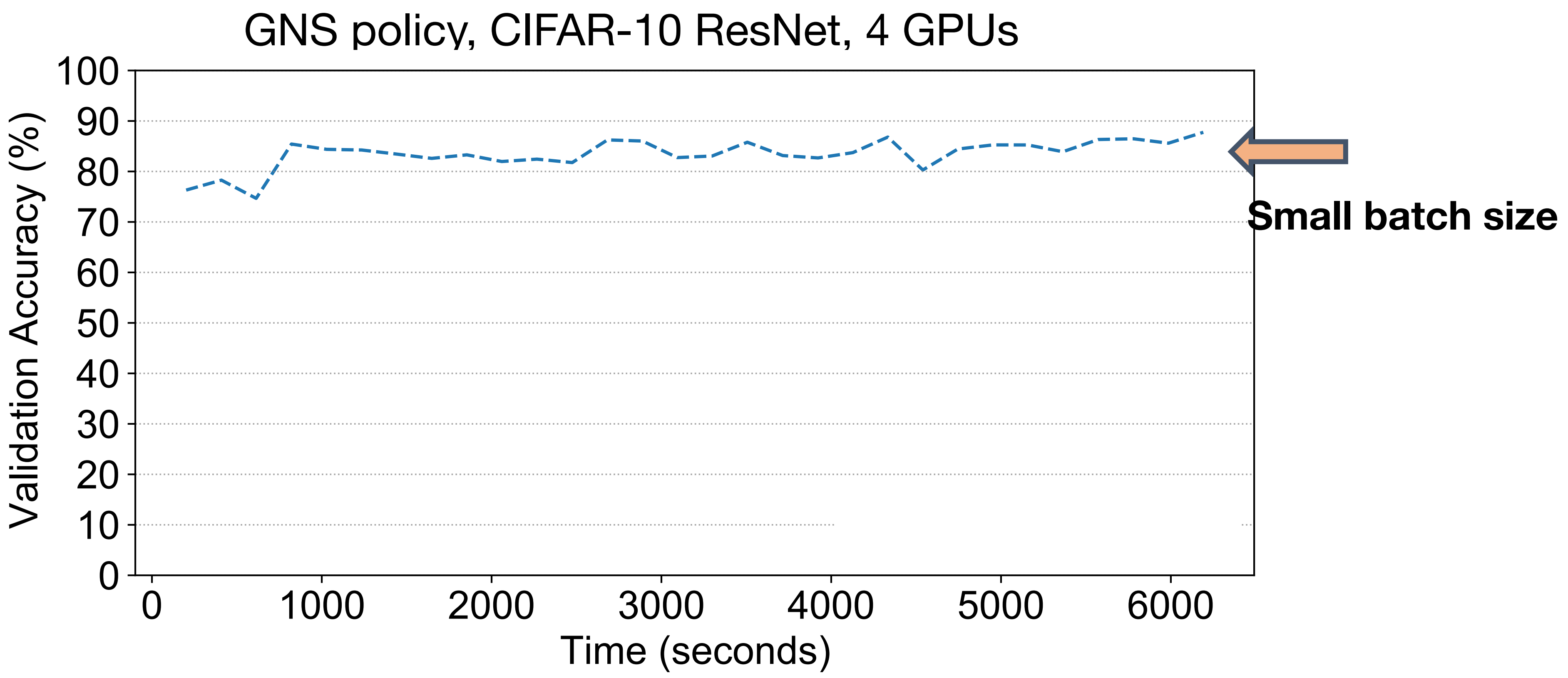
Solution: Wait for collective communication operations to finish before updating parameters



Discard update if communication fails

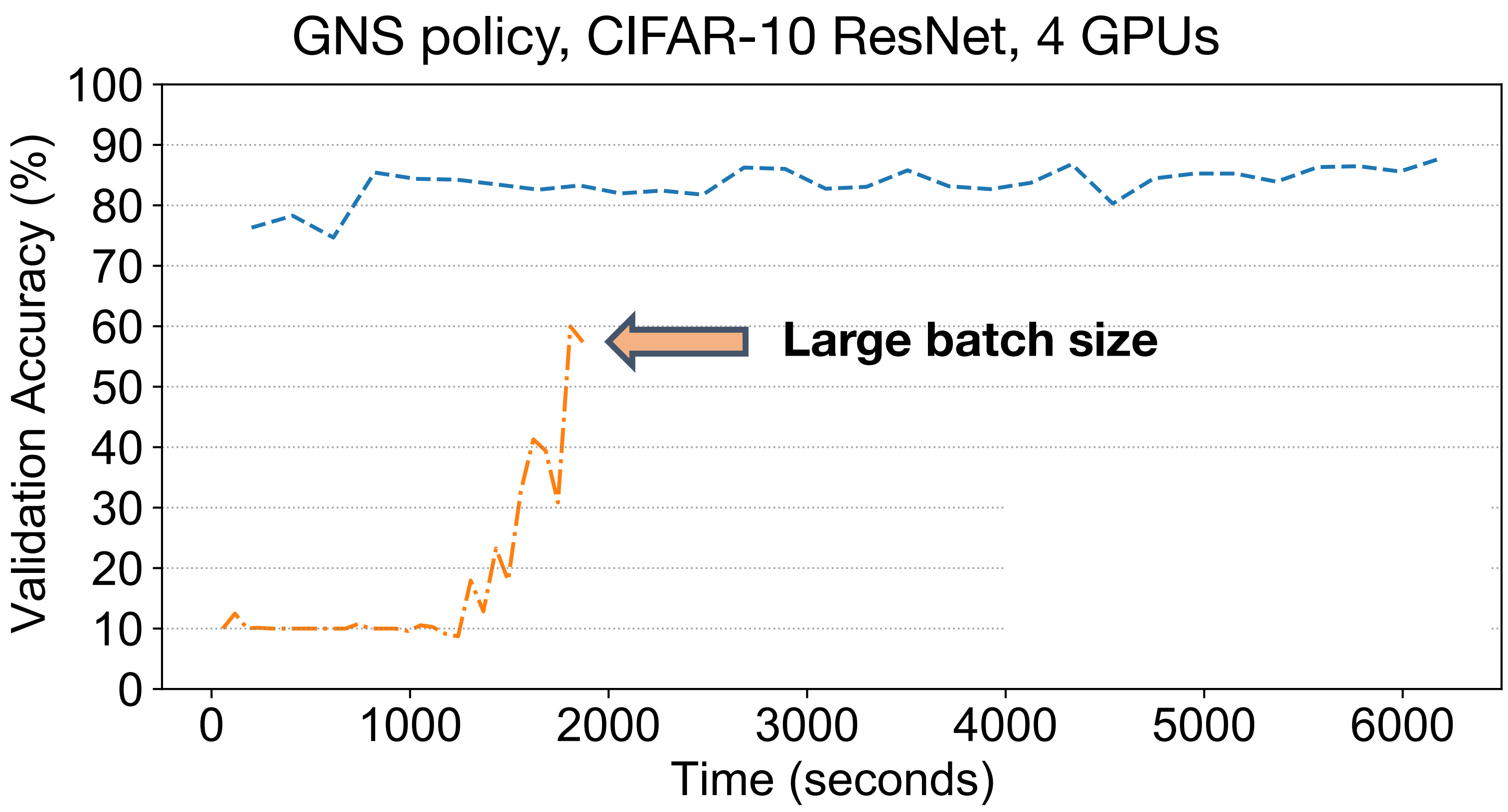
Experimental Evaluation

How Effectively Does KungFu Adapt?



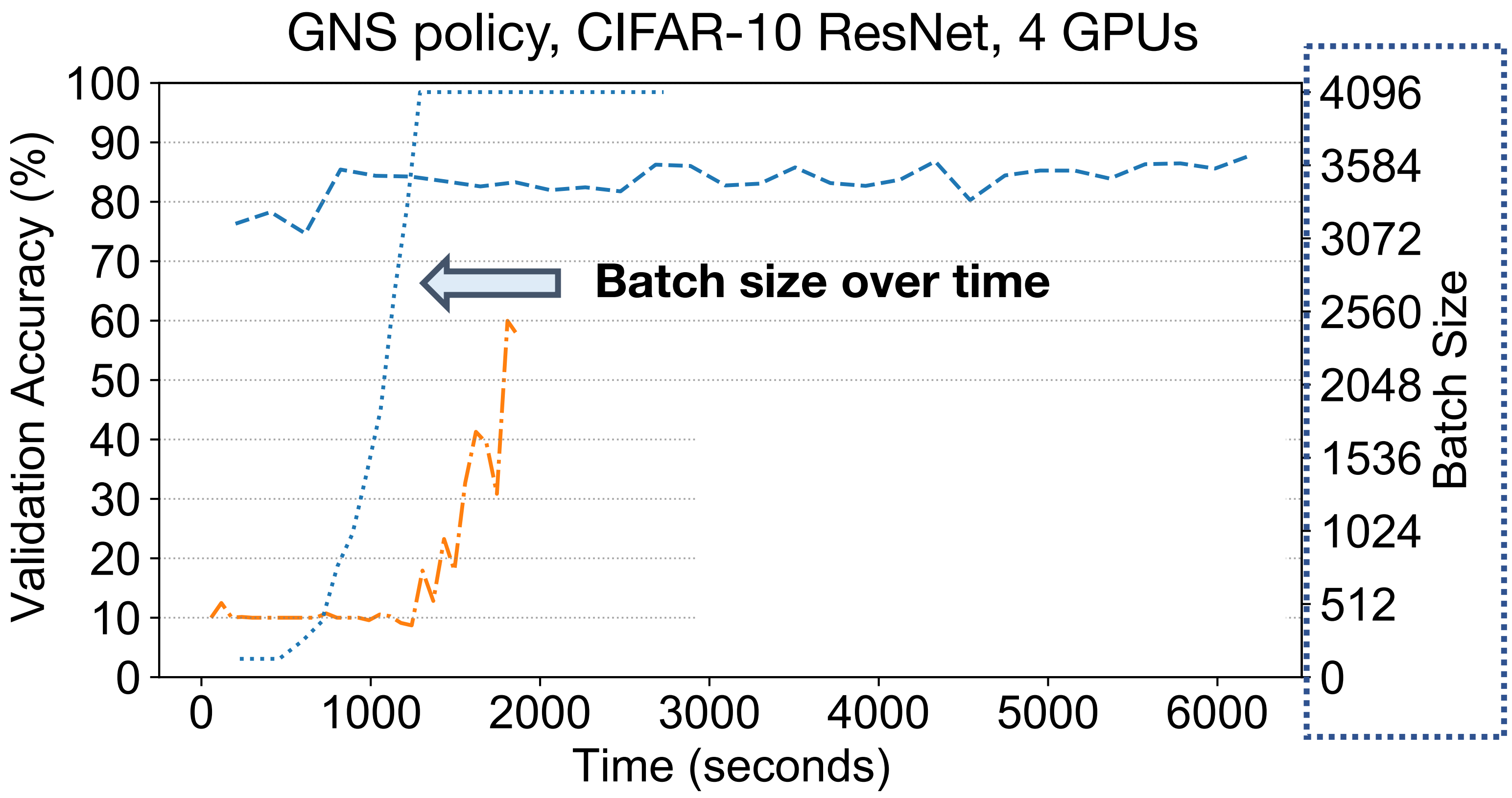
Small batch size reaches high accuracy, but converges slowly

How Effectively Does KungFu Adapt?



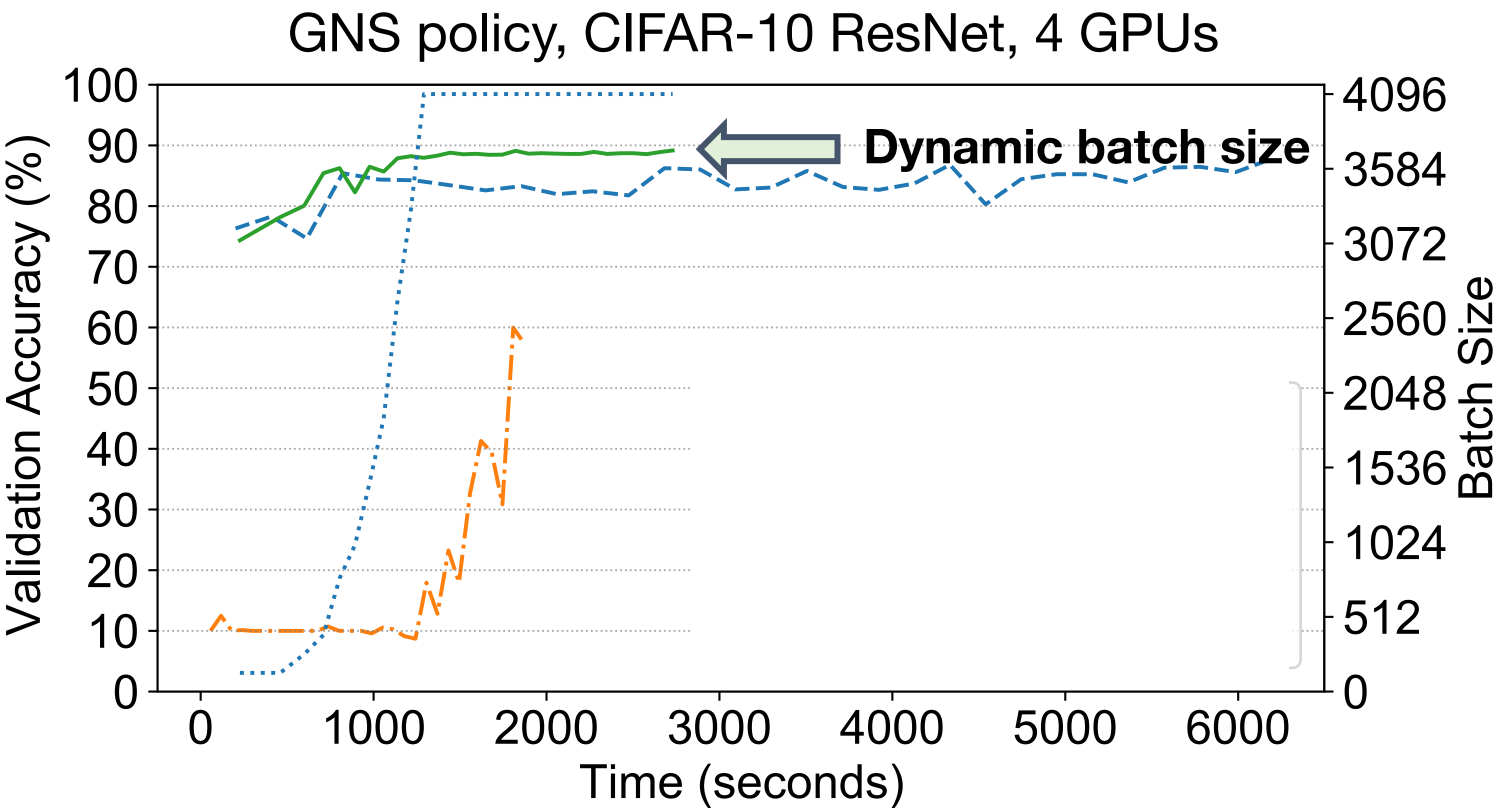
Large batch size finishes quickly, but accuracy suffers

How Effectively Does KungFu Adapt?



GNS predicts how effective batch size should increase during training

How Effectively Does KungFu Adapt?



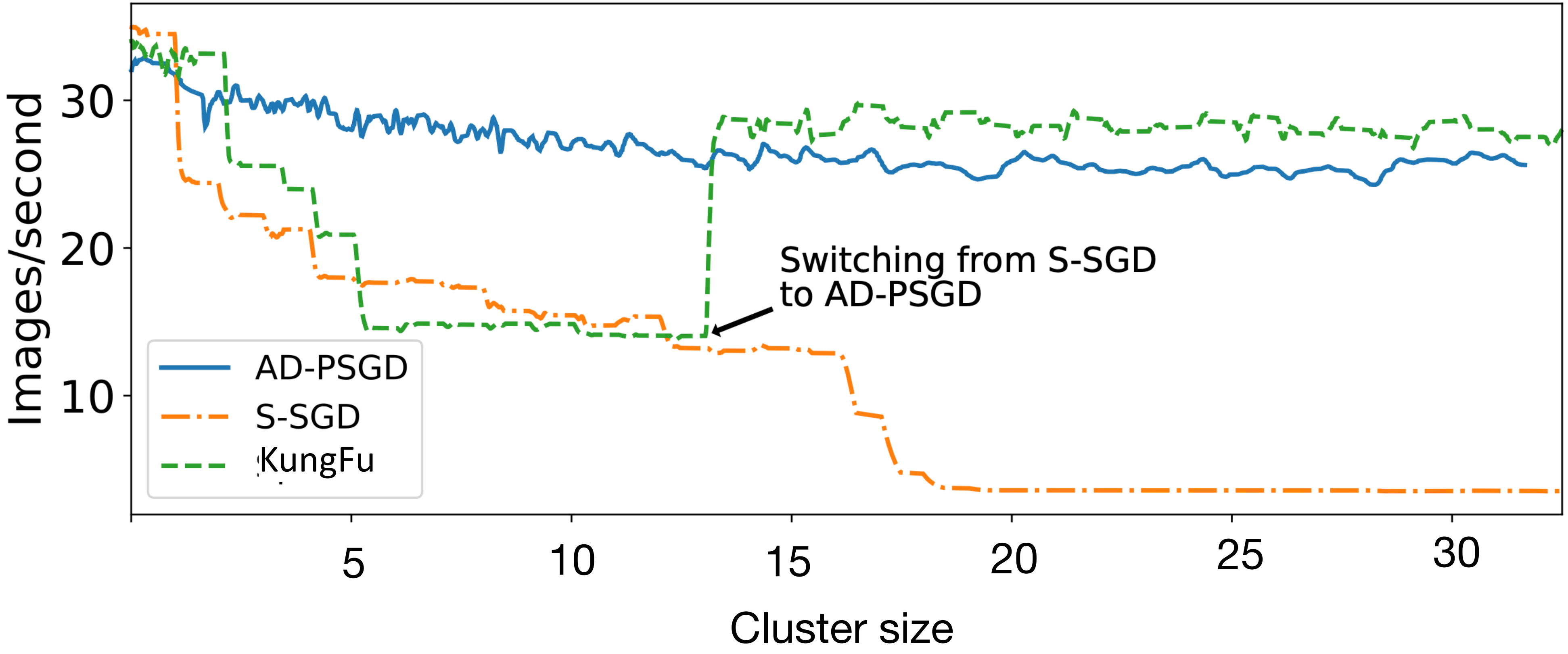
Adaptation Policy has low overhead due to **embedded monitoring**

Does KungFu Adapt to Changing Cluster Sizes?

Cluster:
up to 32 workers

Hardware:
Nvidia K80

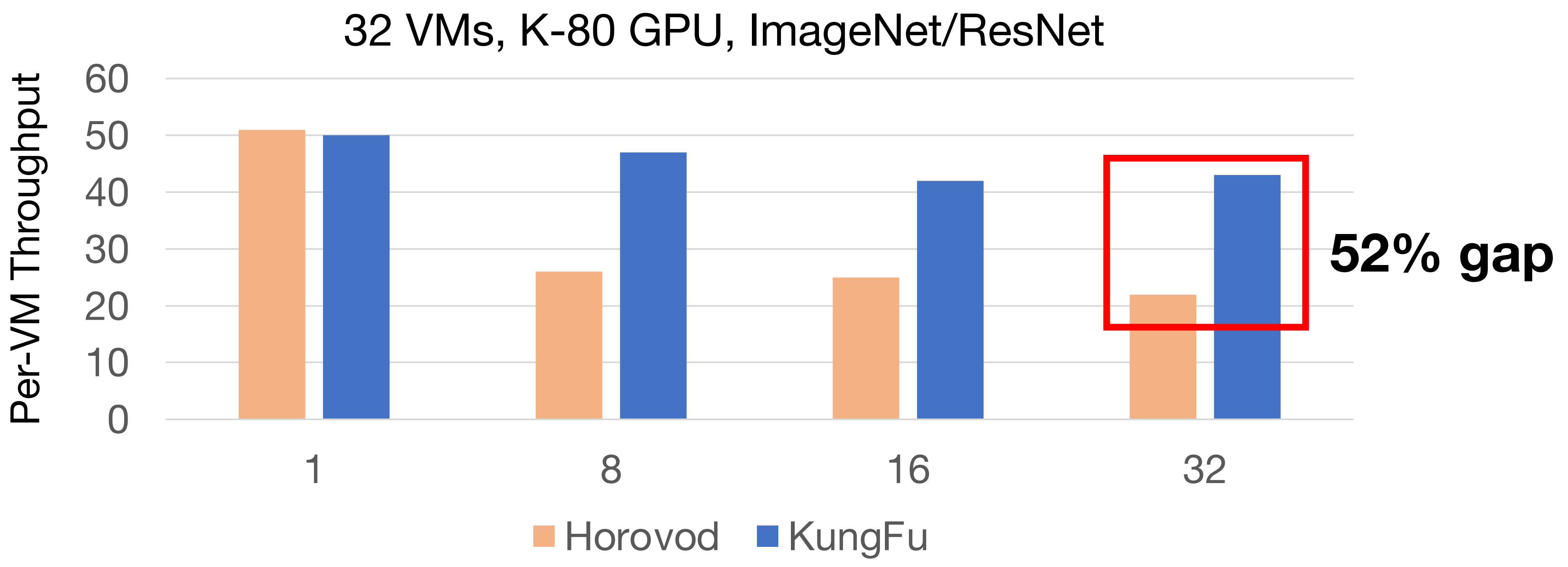
ResNet50/
ImageNet



KungFu switches **synchronisation strategy** based on **cluster size**

What is KungFu's Distributed Performance?

Compare KungFu with state-of-the-art distributed training library (**Horovod**)

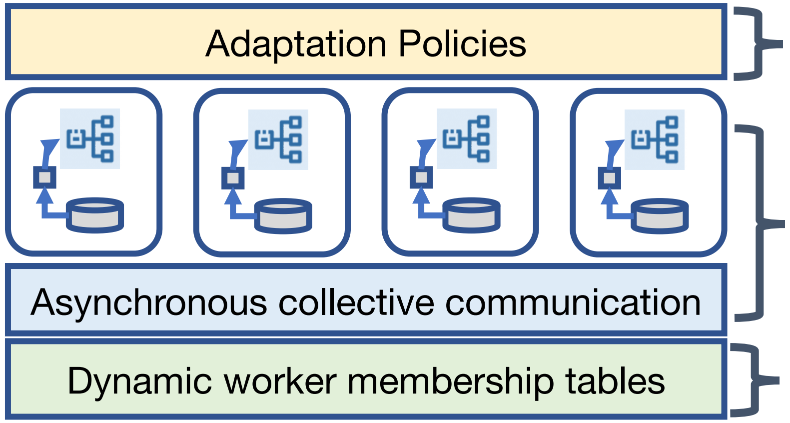


Asynchronous collective communication enables KungFu to scale better

Conclusions: Making Deep Learning Adaptive

Current systems have no unified support for adaptation

KungFu makes distributed deep learning **adaptive**



Decouple adaptation from training program

Take advantage of efficient dataflow execution

Provide powerful distributed primitives

KungFu @ Github
https://github.com/lsds/KungFu

Thank You — Any Questions?

Peter Pietzuch
https://lsds.doc.ic.ac.uk — prp@imperial.ac.uk