Imperial College London

LSDS
Large-Scale Data & Systems Group

THE UNIVERSITY of EDINBURGH

KungFu: Making Training in Distributed Machine Learning Adaptive

Luo Mai [1,2]

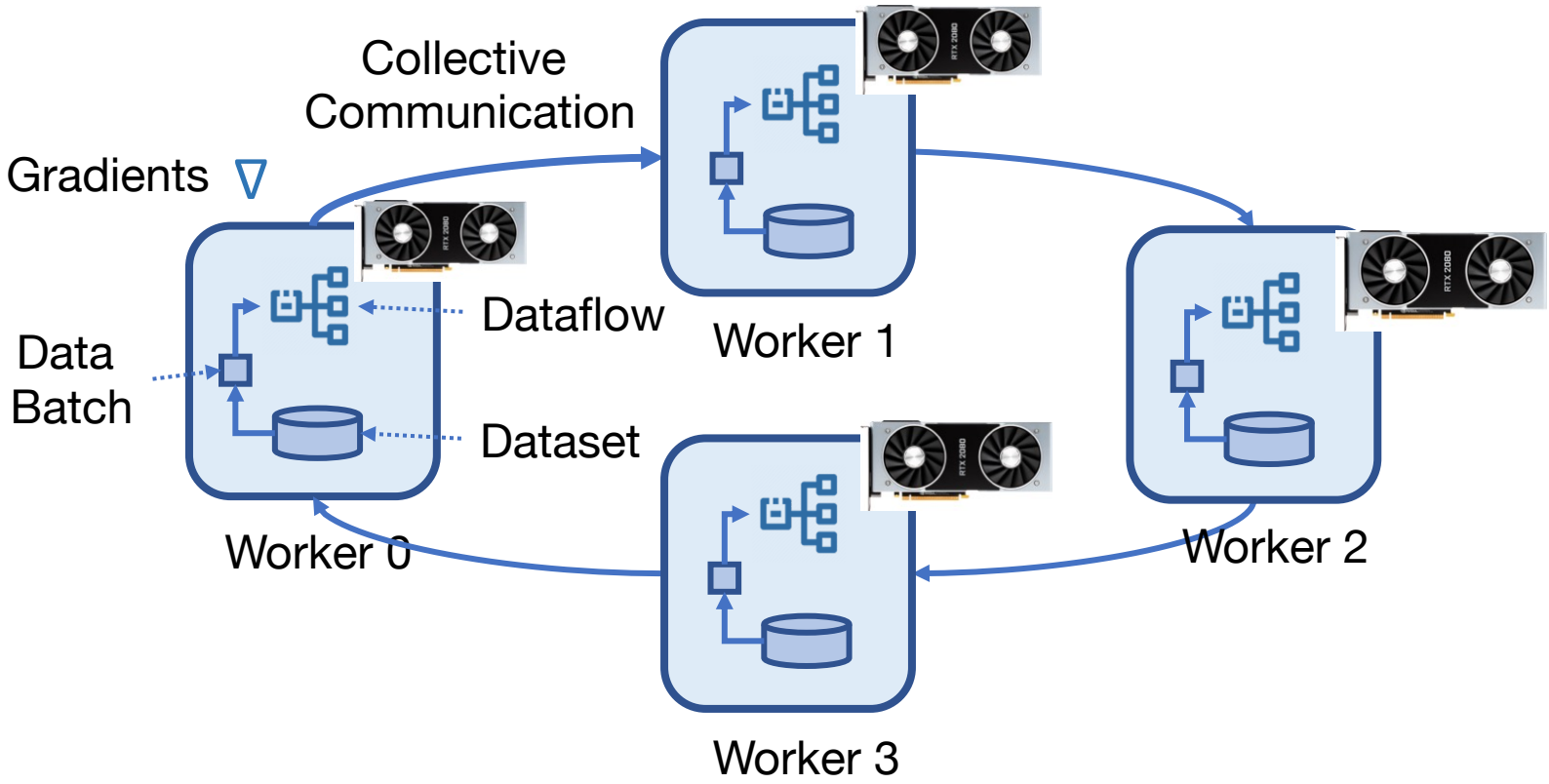
luo.mai@ed.ac.uk

With Guo Li [1], Marcel Wagenlander [1], Konstantinos Fertakis [1], Andrei-Octavian Brabete [1], Peter Pietzuch [1]

Imperial College London [1], University of Edinburgh [2]

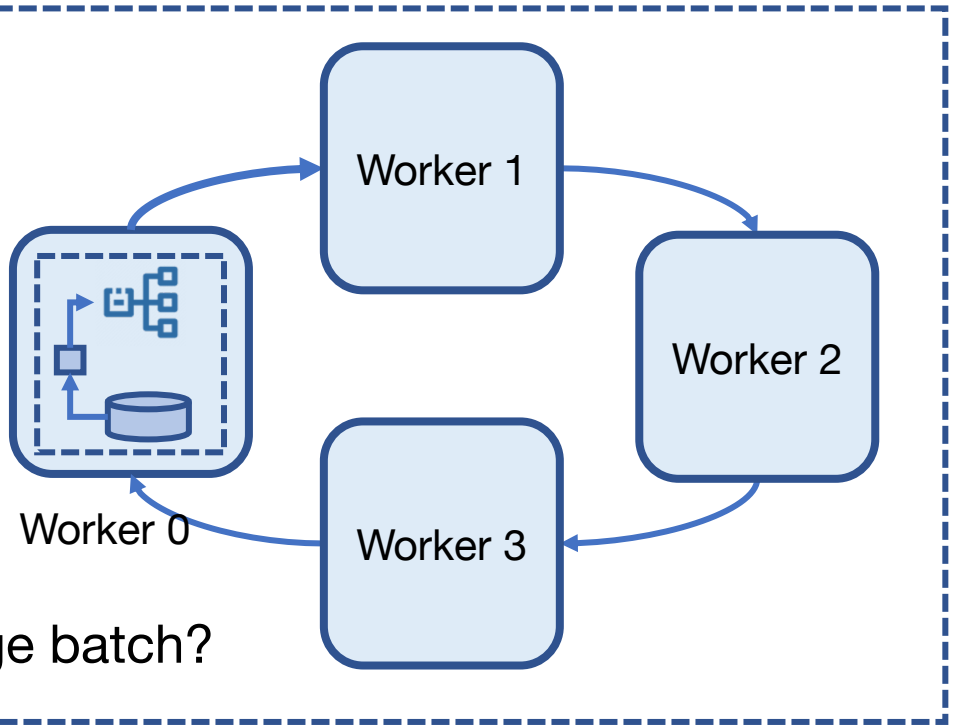
Training in Distributed ML Systems

Distributed training systems are key to combining big data with large models



Parameters in Distributed ML Systems

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



Hyper-parameters
- Batch size
- Learning rate
- …

Small batch or large batch?

System parameters
- Number of workers
- Communication topology
- …

Ring or binary-tree?

Issues with Empirical Parameter Tuning

Examples of empirical parameter tuning	**Issue**
“Change batch size at data 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

Automatic Parameter Adaptation

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

↓

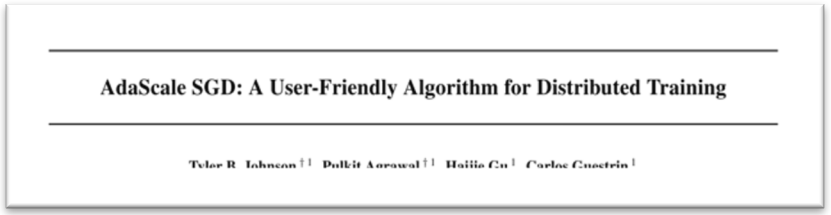
Intuition GNS measures **variation in data batches**

↓

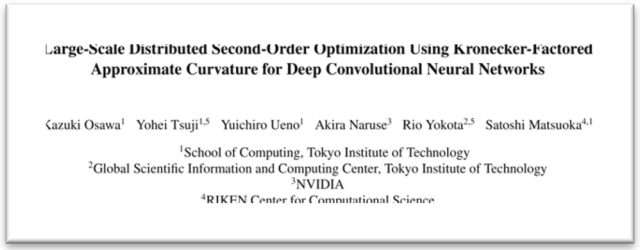
Proposal

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

Proposals for Automatic Parameter Adaptation


AdaScale SGD: A User-Friendly Algorithm for Distributed Training

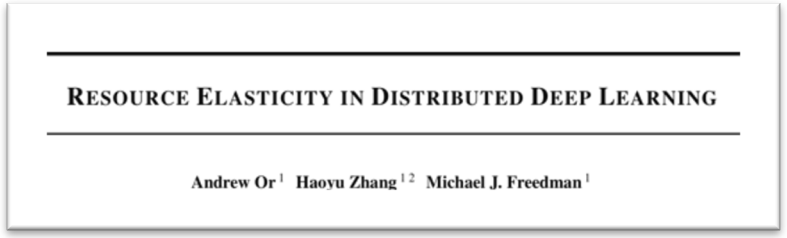
Gradient variance


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]

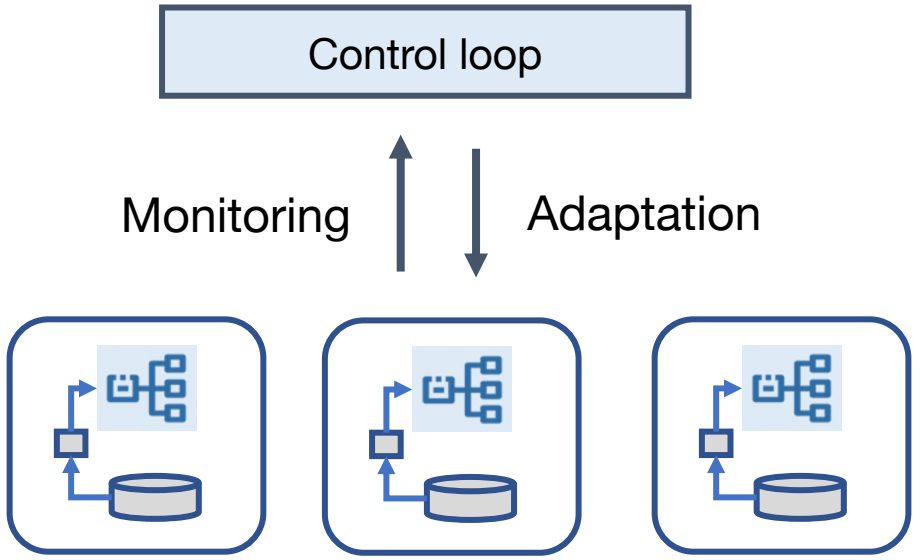
[1]School of Computing, Tokyo Institute of Technology
[2]Global Scientific Information and Computing Center, Tokyo Institute of Technology
[3]NVIDIA
[4]RIKEN Center for Computational Science

Gradient second-order metrics


RESOURCE ELASTICITY IN DISTRIBUTED DEEP LEARNING

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

Worker performance metrics



Control loop monitors training workers and use monitored metric to set parameters

Open Challenges

Can we design a distributed ML system that supports adaptation?

Design challenges:

- **How to support different types of adaptation?**
- **How to adapt based on large volume of monitoring data?**
- **How to change parameters of stateful workers?**

Existing Approaches for Adaptation

1. No unified mechanisms for adaptation

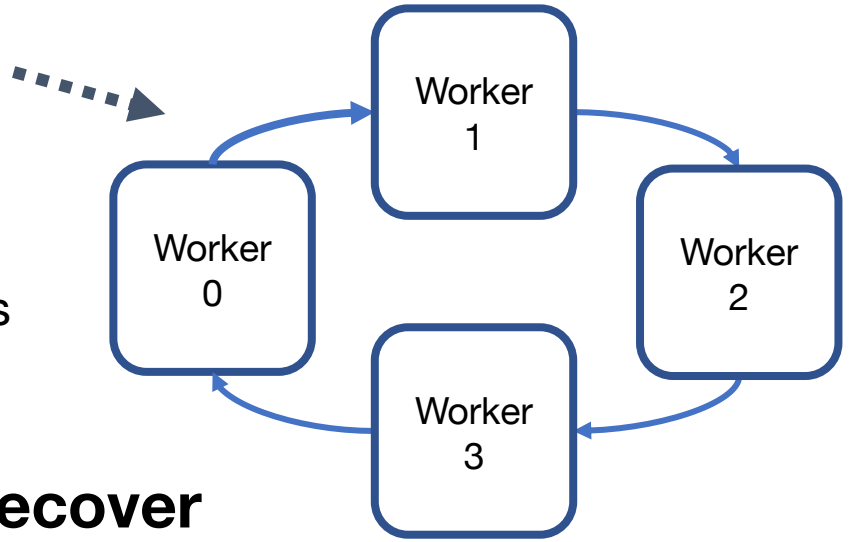
AutoScaling [MLSys'20]

Horovod Elastic

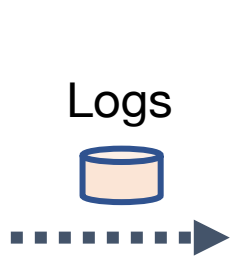
PyTorch Elastic

TensorFlow Elastic

Custom adaptation frameworks without generic APIs

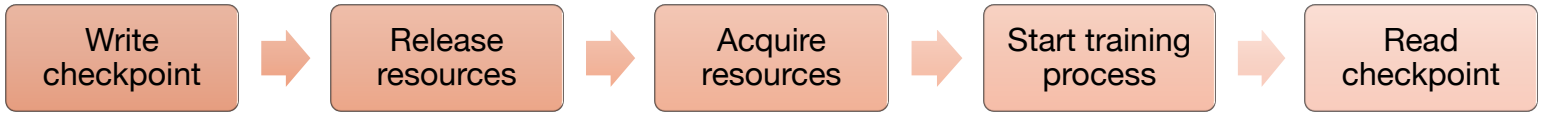


2. Processing of monitoring data offline



Log analytics libraries

3. Checkpoint-and-recover



Many adaptation steps

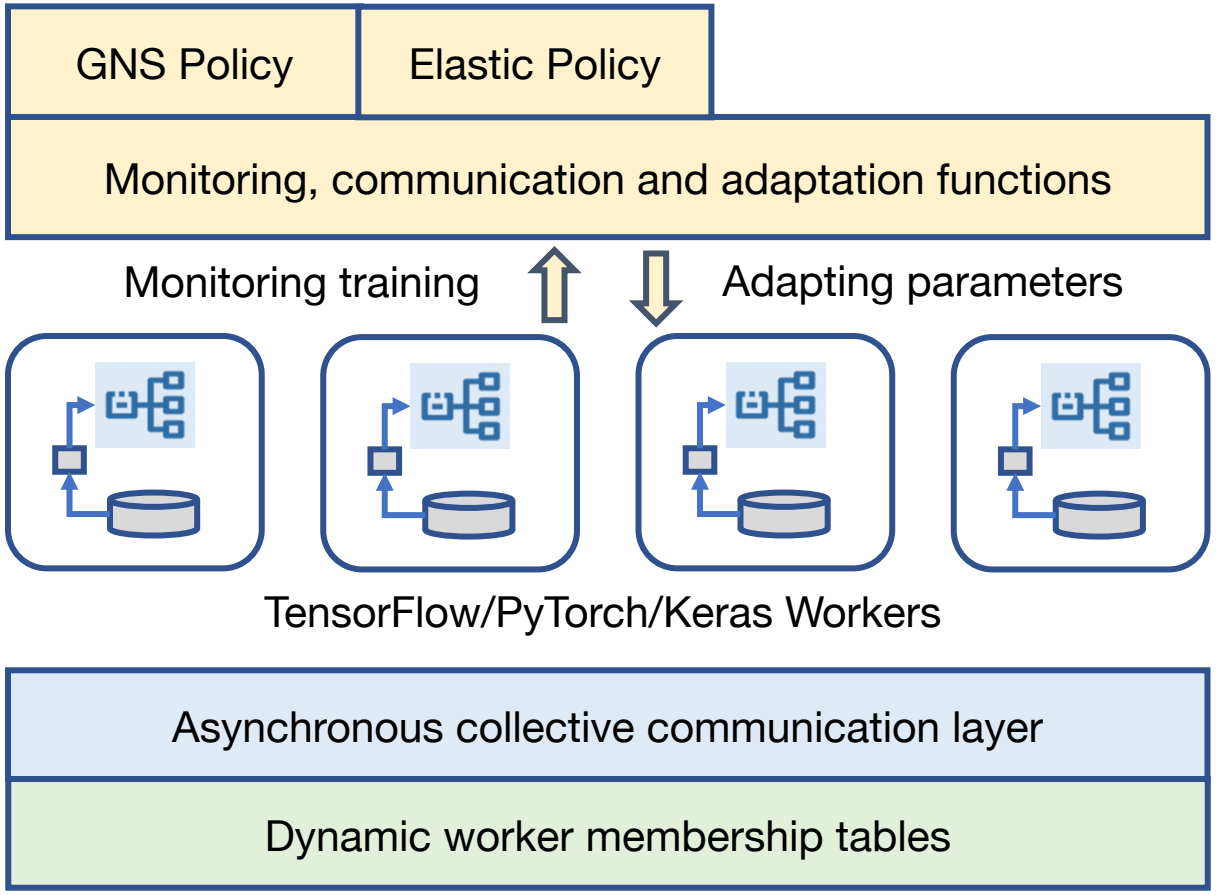
KungFu Overview

Contributions

1. Adaptation policies

2. Embedding monitoring operators inside dataflow

3. Distributed mechanisms for parameter adaptation



Supporting **different types of adaptation**

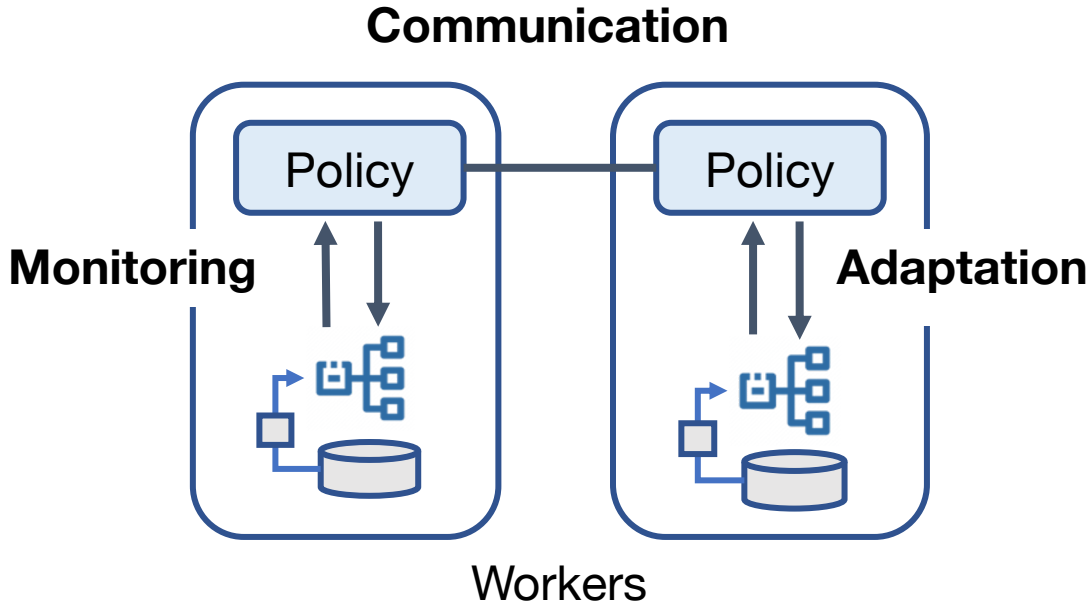
Processing **large volume of monitoring data**

Consistent adaptation for stateful workers

Contribution 1
Adaptation Policies

Adaptation Policies

Goal: Express adaptation over control loop for parameters



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

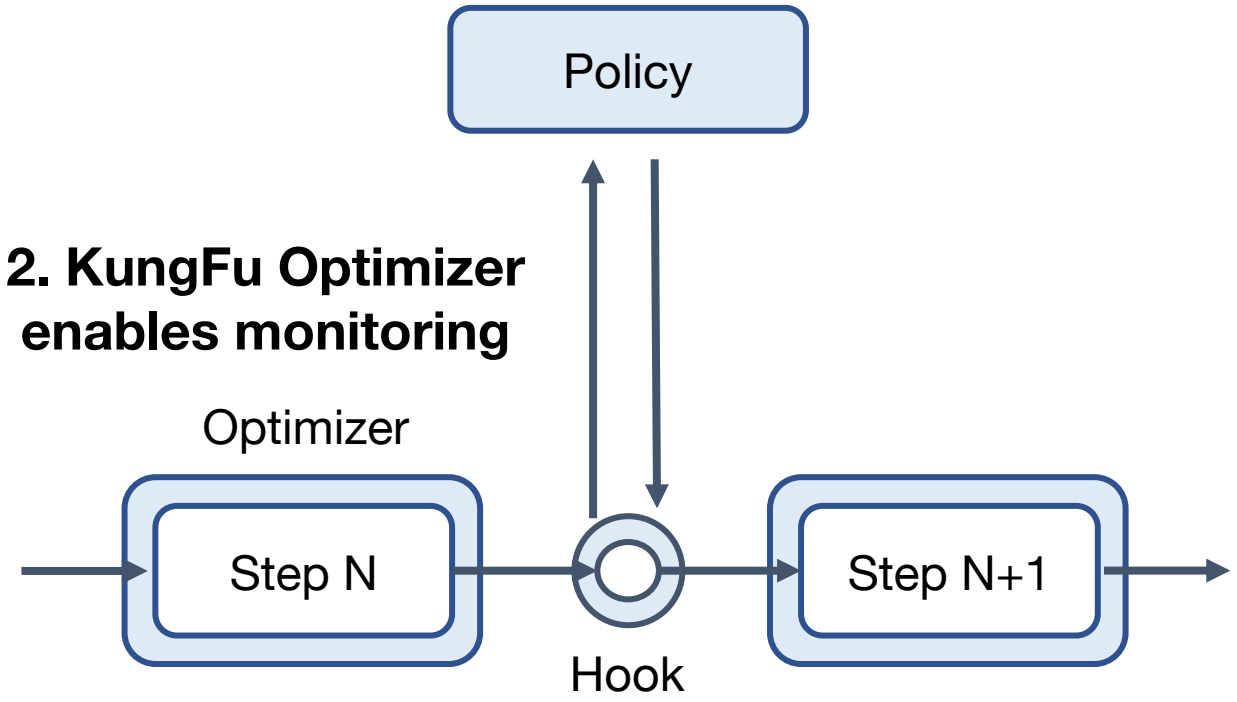
1. Adaptation logic in policy functions

```
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)
```

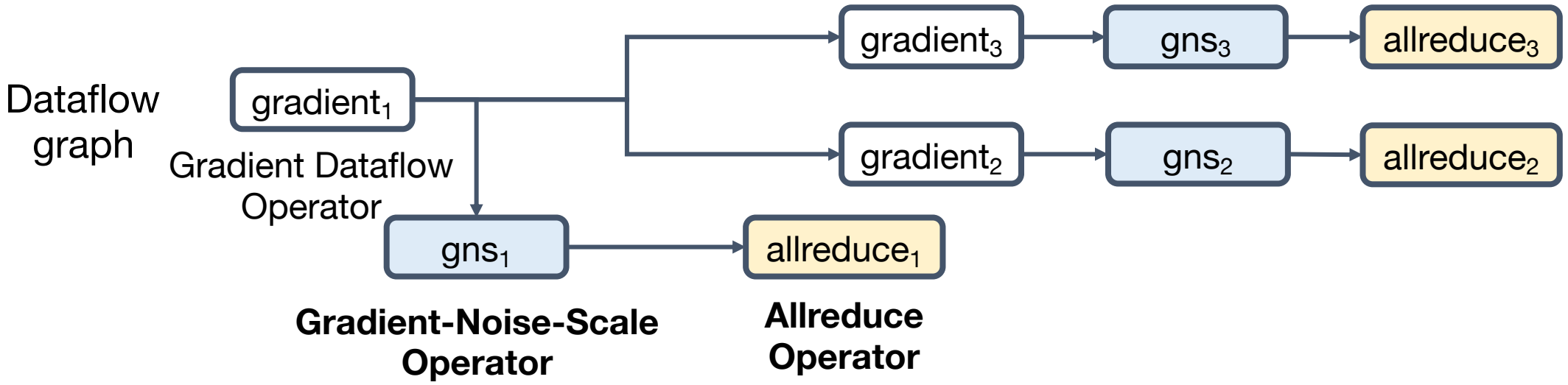


3. KungFu Hook enables the policy

Contribution 2
Embedding Monitoring Inside Dataflow

Embedding Monitoring Inside Dataflow

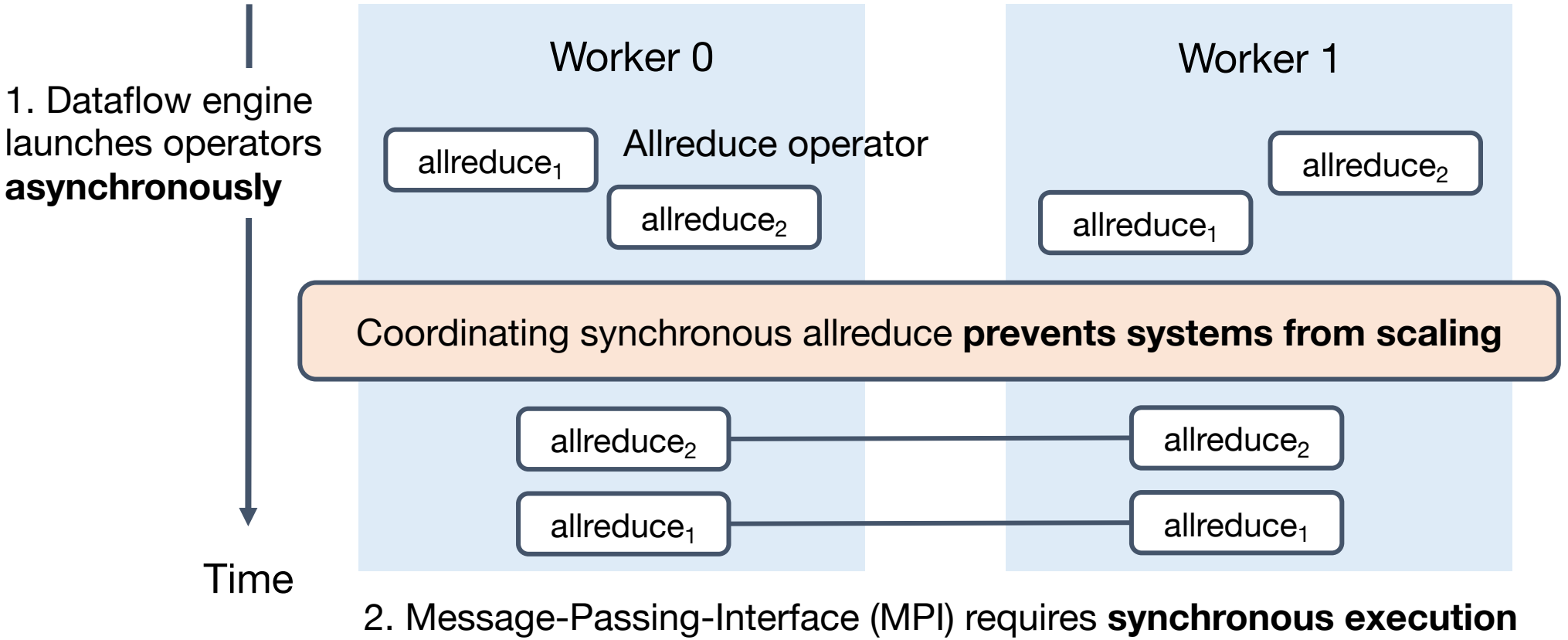
Problem: High monitoring cost reduces adaptation benefit
Idea: Improve efficiency by **adding monitoring operators to dataflow graph**



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

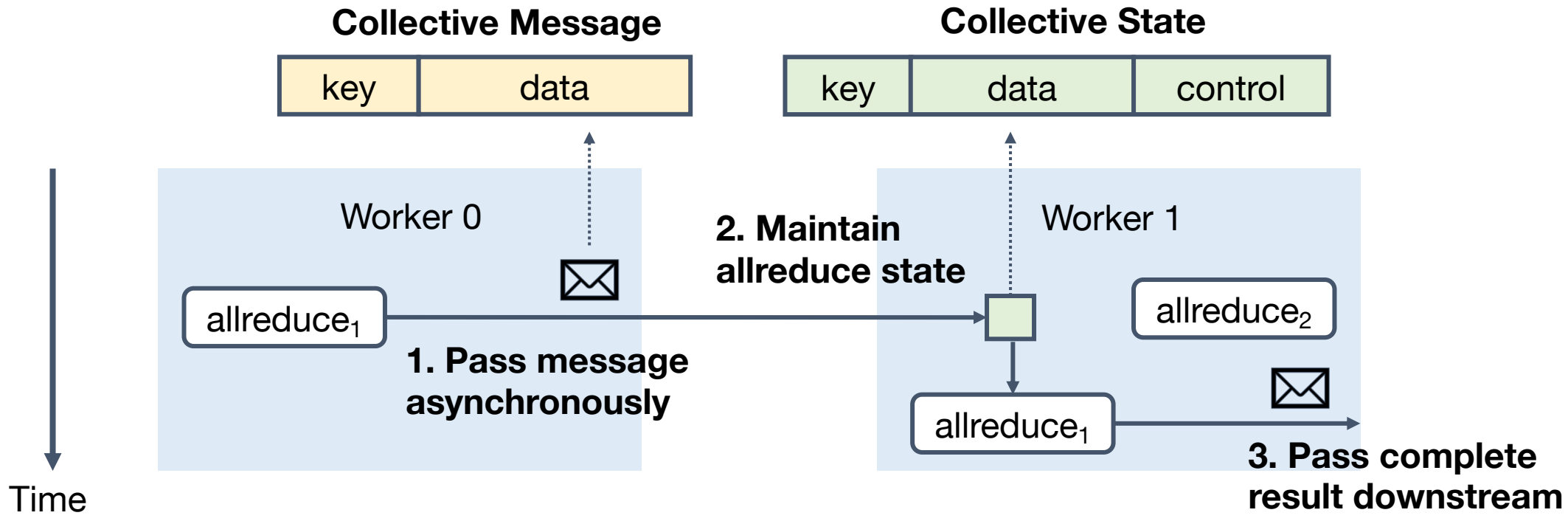
Challenges of Dataflow Collective Communication

Problem: Collective communication reduces dataflow performance



Making Collective Communication Asynchronous

Idea: Use asynchronous collective communication

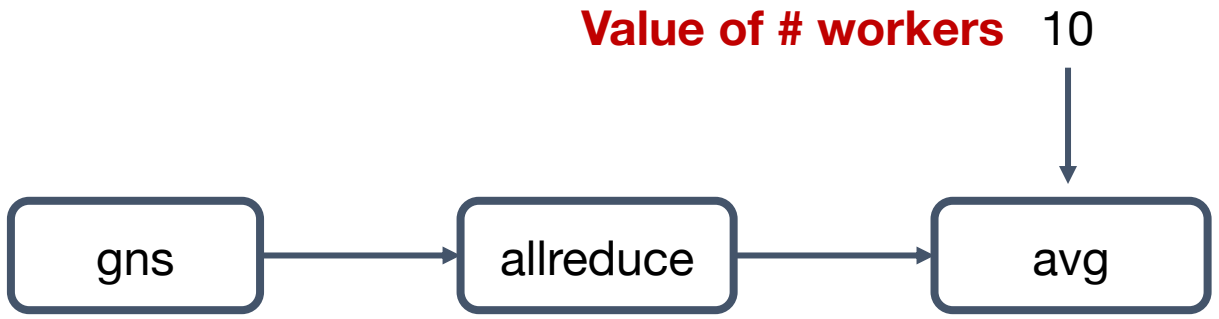


No need for coordination in asynchronous collective communication

Contribution 3
Distributed Mechanisms for Parameter Adaptation

Issues When Adapting System Parameters

Problem: Parameter adaptation affects state consistency



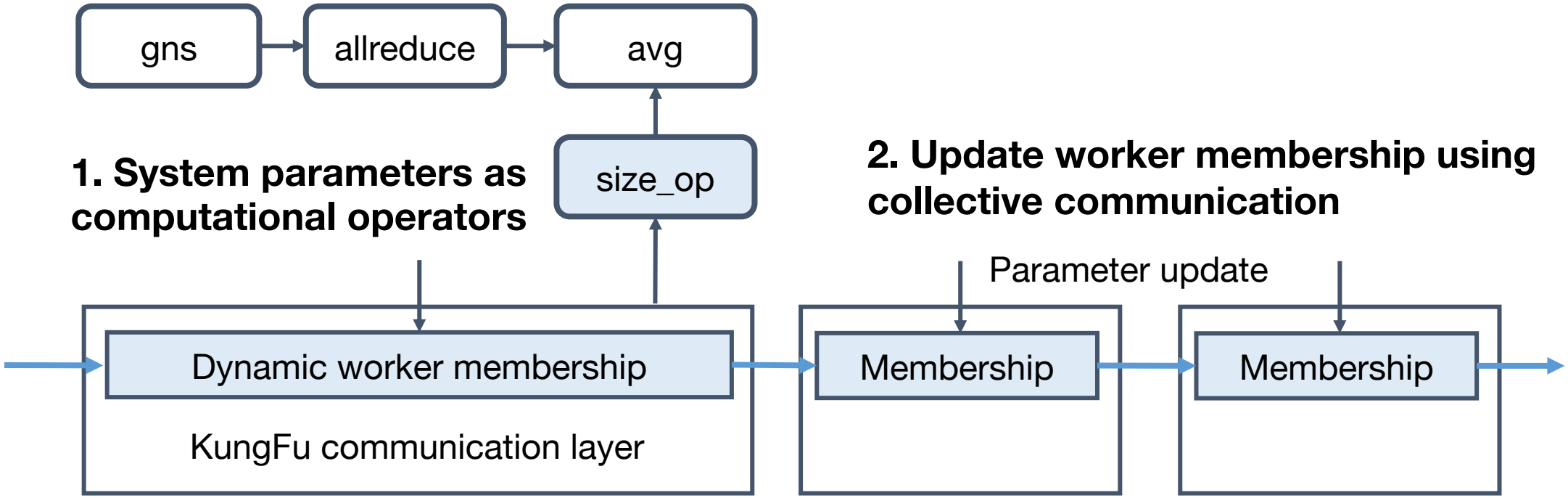
Dataflow for averaging GNS

Other system parameters
- Worker ranks
- Communication topology
- …

Adapting system parameters therefore often requires system restart

Distributed Mechanism for Parameter Adaptation

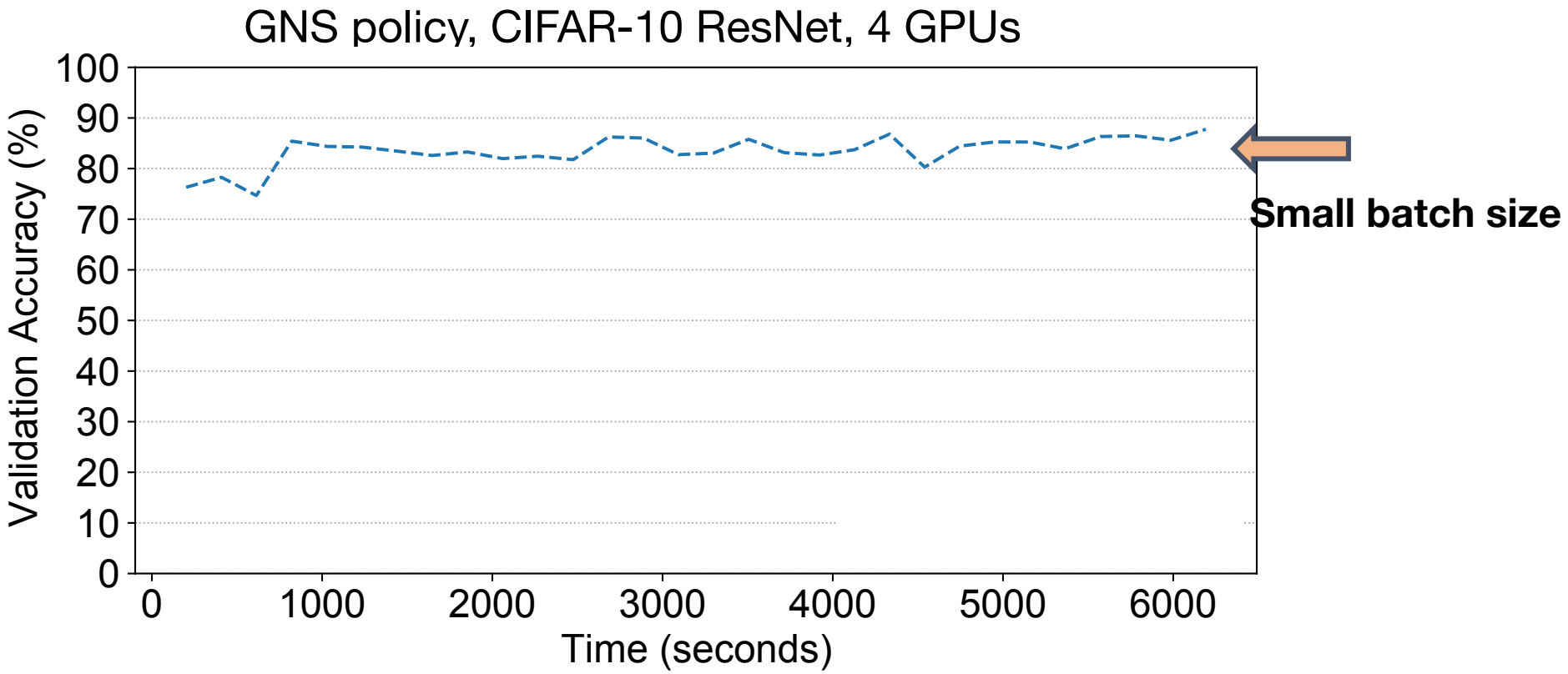
Idea: Decouple system parameters with dataflow state



Online parameter adaptation is consistent and fast

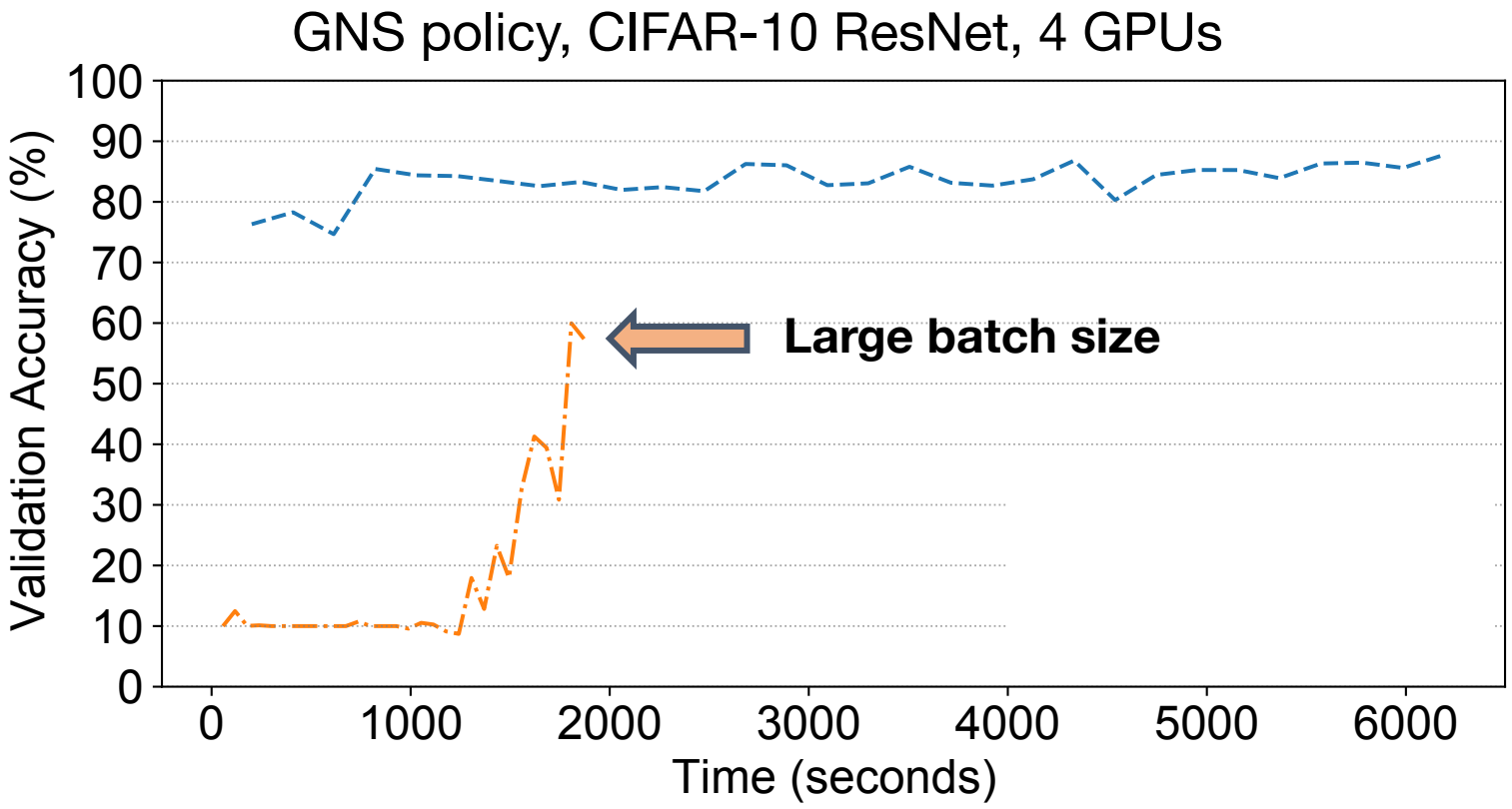
Experimental Evaluation

How Effectively Does KungFu Adapt?



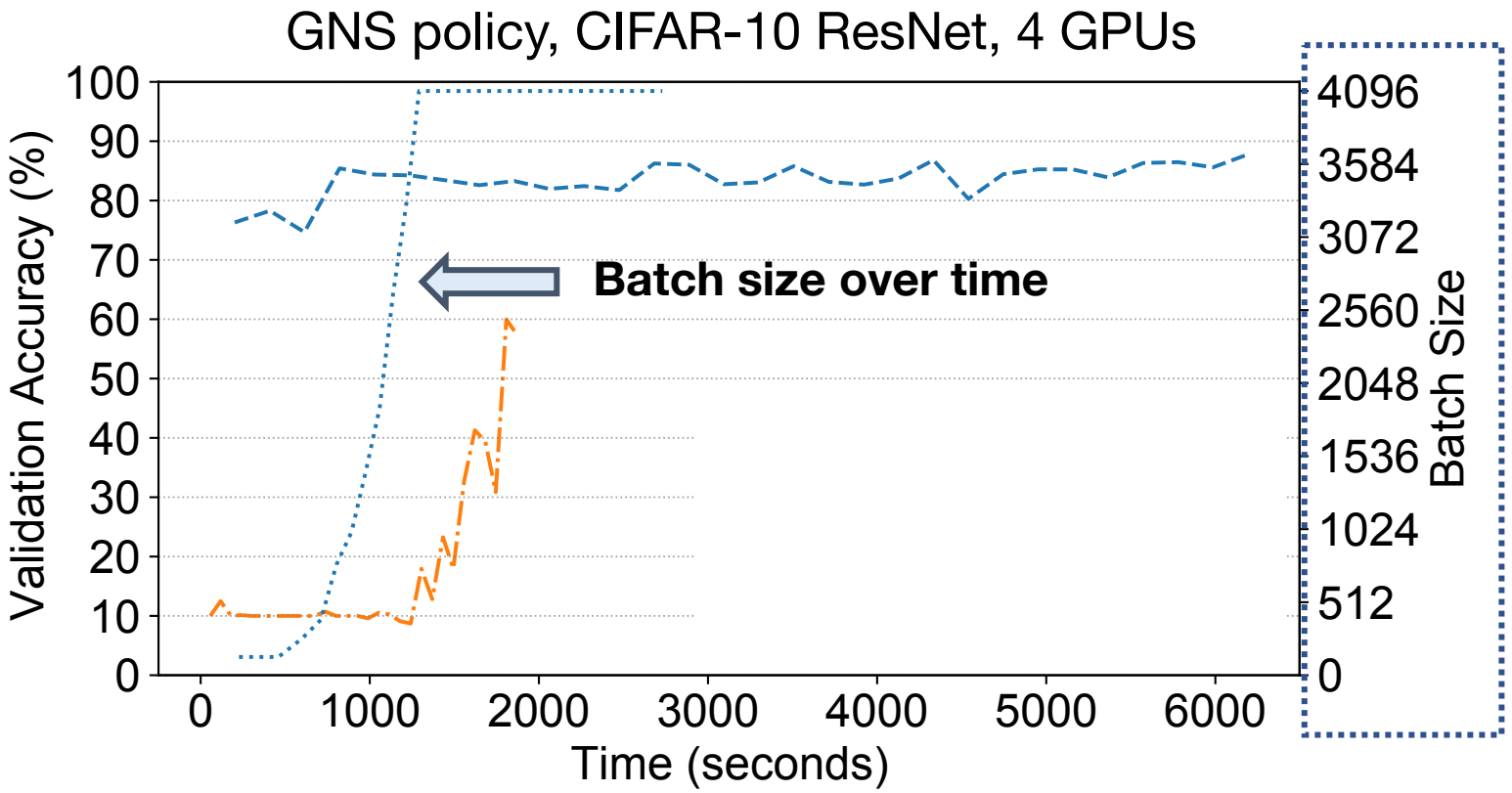
Small batch size reaches high accuracy, but converges slowly

How Effectively Does KungFu Adapt?



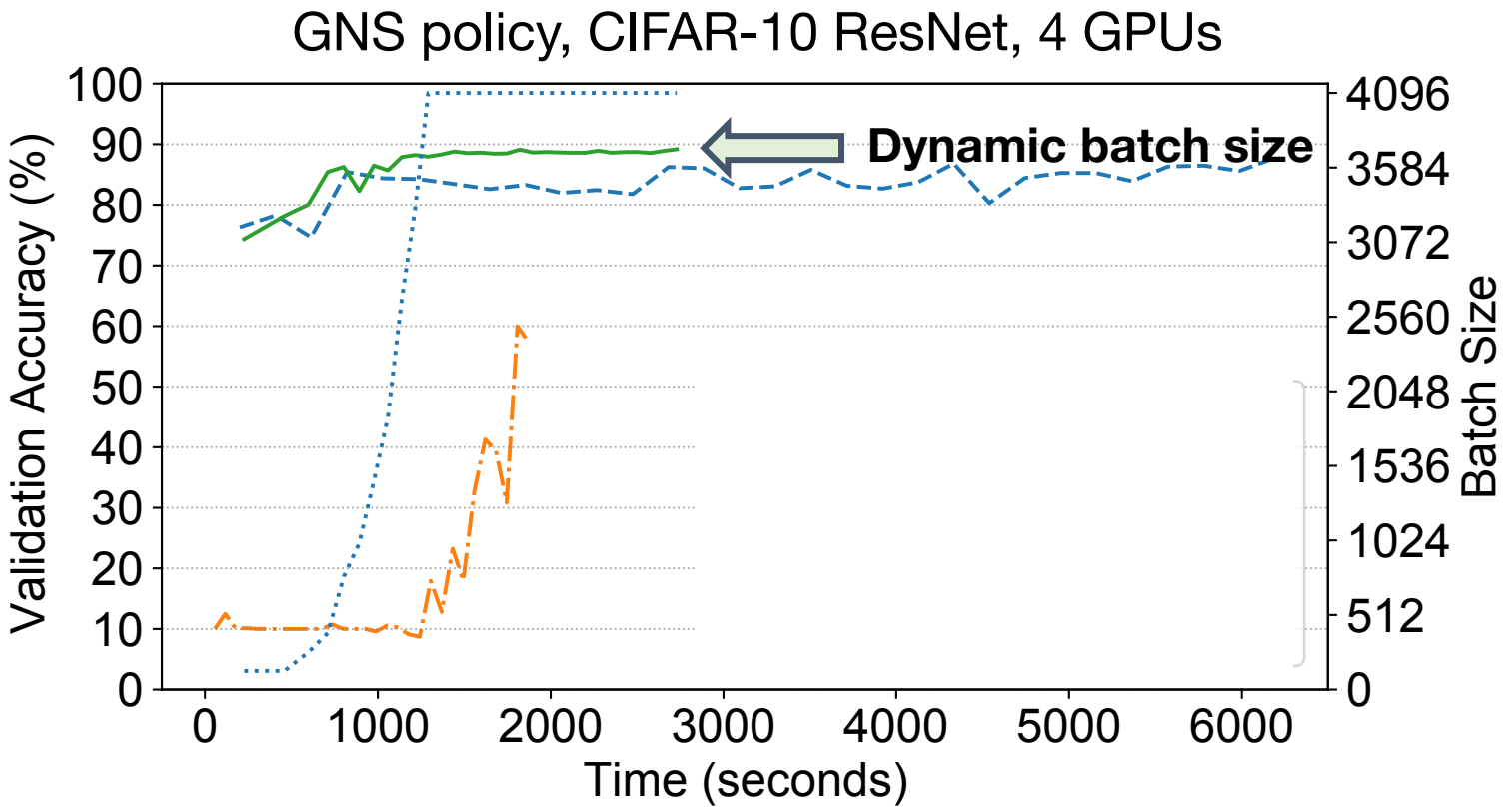
Large batch size finishes quickly, but accuracy suffers

How Effectively Does KungFu Adapt?



GNS predicts the effective batch size should increase during training

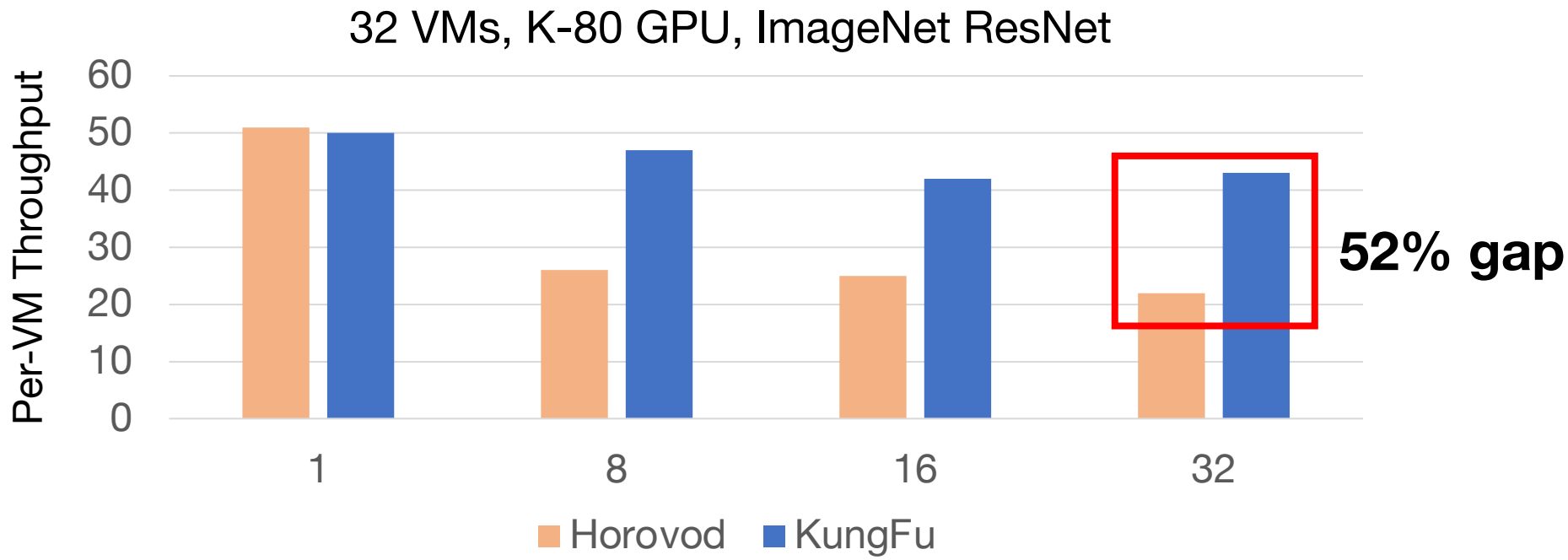
How Effectively Does KungFu Adapt?



Embedded monitoring and **online adaptation** are important to Adaptation Policies

What is KungFu's Distributed Performance?

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

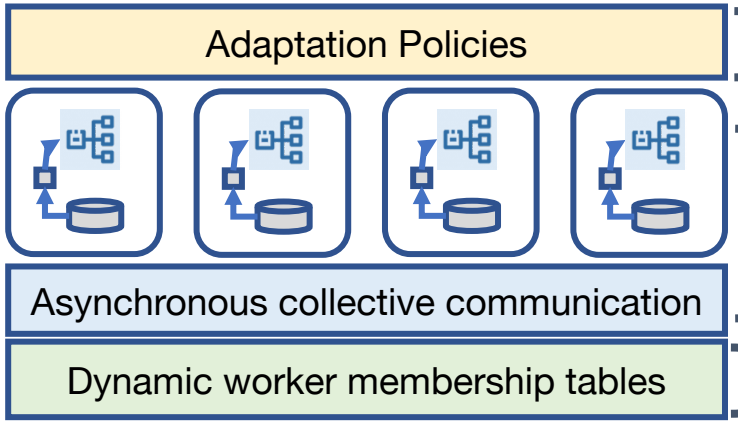


Asynchronous collective communication enables KungFu to scale better

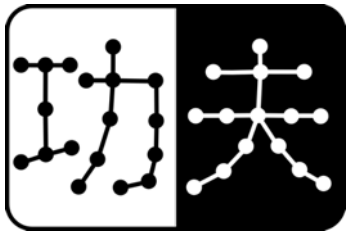
Conclusions: KungFu

KungFu makes distributed machine learning **adaptive**

- Current systems have no unified mechanism for adaptation



1. **Adaptation policies** that realise complex adaptation
2. **Embedding monitoring inside dataflow**
3. **Distributed mechanisms for consistent adaptation**



KungFu @ Github

https://github.com/lsds/KungFu