Abstracting Systems Challenges in Distributed Deep Learning

Andrew Or

Princeton University
Advised by Michael J. Freedman
Model Sizes are Increasing

- MegatronLM 8.3B
- BERT-LARGE 340M
- Turing-NLG 17B
- GPT-3 175B

# Batch Sizes are Increasing

<table>
<thead>
<tr>
<th></th>
<th>Batch Size</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goyal et al. [3]</td>
<td>8K</td>
<td>Pascal GPU x 256</td>
</tr>
<tr>
<td>Smith et al. [42]</td>
<td>16K</td>
<td>Full TPU Pod</td>
</tr>
<tr>
<td>Codreanu et al. [43]</td>
<td>32K</td>
<td>KNL x 1024</td>
</tr>
<tr>
<td>You et al. [41]</td>
<td>32K</td>
<td>KNL x 2048</td>
</tr>
<tr>
<td>Akiba et al. [38]</td>
<td>32K</td>
<td>Pascal GPU x 1024</td>
</tr>
<tr>
<td>Jia et al. [18]</td>
<td>64K</td>
<td>Pascal GPU x 1024</td>
</tr>
<tr>
<td>Jia et al. [18]</td>
<td>64K</td>
<td>Pascal GPU x 2048</td>
</tr>
<tr>
<td>Mikami et al. [44]</td>
<td>68K</td>
<td>Volta GPU x 2176</td>
</tr>
</tbody>
</table>

Training ResNet-50 on ImageNet for 90 epochs
Heterogeneity is Increasingly Common

Job 1

Job 2

Job 3

Narayanan et al. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads
High resource req.
Tune once per cluster
Static allocation
Homogeneous allocation

Real world
Wide range of users
Tuning is cumbersome
Dynamic availability
Heterogeneous clusters
Deep learning **Models** are tightly coupled with the underlying **Hardware**
Model + Hyperparameters
- ResNet
- BERT
- Transformer
- Batch size
- Learning rate
- Momentum

Deep Learning Frameworks
- mxnet
- PyTorch
- TensorFlow

Physical Resources
- GPU, TPU, FPGA
Abstracting Systems Challenges in Distributed Deep Learning

Resource Elasticity in Distributed Deep Learning
Andrew Or, Haoyu Zhang, Michael J. Freedman. MLSys 2020.

VirtualFlow: Decoupling Deep Learning Models from Underlying Hardware
Andrew Or, Haoyu Zhang, Michael J. Freedman. (Under submission)
Resource Elasticity in Distributed Deep Learning

VirtualFlow: Decoupling Deep Learning Models from Underlying Hardware

- Reproducibility
- Experimentation
- Elasticity Exploration
- Hyperparameter Exploration
- Elasticity Convergence Guarantees
- Heterogeneous Training

- Elasticity Mechanisms
- Straggler Mitigation
- Autoscaling Heuristics
Data Parallel, Synchronous Training

Ring reduce

Batch 1  Batch 2  Batch 3

... x 100
Resource Elasticity in Distributed Deep Learning

Andrew Or, Haoyu Zhang*, Michael J. Freedman

Princeton University, *Google AI
Users rely on manual trial-and-error process to find resource efficient cluster size
Manual Trial-and-Error Resource Allocation

Requires expertise 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
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**
Resource Elasticity is Not a New Idea

<table>
<thead>
<tr>
<th>Distributed computing</th>
<th>Cloud services</th>
<th>Cluster management</th>
<th>Distributed deep learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Spark</td>
<td>AWS</td>
<td>Hadoop YARN</td>
<td>Apache Mesos</td>
</tr>
<tr>
<td>Hadoop MapReduce</td>
<td>Azure</td>
<td>Apache Storm</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>Presto</td>
<td>Google Cloud</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
State-of-the-art

**Autoscaling:** Spark, MapReduce, Kubernetes, AWS, Azure etc.
- Or et al. Apache Spark Dynamic Resource Allocation
- Dean et al. MapReduce: Simplified Data Processing on Large Clusters
- AWS Auto Scaling: https://aws.amazon.com/autoscaling/

**Elasticity in ML:** Litz, Proteus, Declarative ML elasticity
- Huang et al. Resource elasticity for large-scale machine learning
- Harlap et al. Addressing the straggler problem for iterative convergent parallel ML
- Qiao et al. Litz: Elastic framework for high-performance distributed machine learning
- Harlap et al. Proteus: agile ml elasticity through tiered reliability in dynamic resource markets

**GPU cluster schedulers:** Gandiva, Optimus, Tiresias, Themis
- Peng et al. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters
- Gu et al. Tiresias: A GPU Cluster Manager for Distributed Deep Learning
- Mahajan et al. THEMIS: Fair and Efficient GPU Cluster Scheduling
Hurdle #1: Static Model Graph

Communication ops are embedded in these graphs

Synchronization primitives assume fixed # GPUs

E.g. TensorFlow’s MultiWorkerMirroredStrategy

[Abadi et al., 2015]
Hurdle #2: How to Scale the Batch Size?

1) Fix per GPU batch size, vary global batch size
   - Preserves per GPU efficiency
   - Large batch sizes may compromise convergence behavior
     [Keskar et al., 2016; Goyal et al., 2017; Hoffer et al., 2017]

2) Fix global batch size, vary per GPU batch size
   - Preserves convergence behavior
   - Sacrifices per GPU efficiency and overall performance
Hurdle #3: 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

Deep learning: resource usage is *cyclical and consistent*
Why is Resource Elasticity Not Adopted Yet?

**Hurdle #1:** Static model graph

**Hurdle #2:** How to scale the batch size?

**Hurdle #3:** Lack of applicable scaling heuristics
Hurdle #1: Static Model Graph

Communication operations are **statically connected** across accelerators

Extract communication ops, replace with **narrow-waist, black-box function**
Hurdle #1: Static Model Graph

Communication operations are *statically connected* across accelerators

Extract communication ops, replace with *narrow-waist, black-box function*

Give workers the *illusion of local training*, replace function when scaling
Hurdle #2: How to Scale the Batch Size?

Temp solution: place upper bound on global batch size (e.g. 4096)

Finding this upper bound is an open problem

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

Addressed in VirtualFlow
Hurdle #3: Lack of Applicable Scaling Heuristics

Two new scaling heuristics:

*Throughput scaling efficiency*

*Utility vs cost*

... more details in the paper
Autoscaling Engine for Distributed Deep Learning

Execution engine
- Forward / backward pass
- Raw gradients ↔ Synced gradients
- Parameter synchronization

Autoscaling engine
- Scaling heuristics
- Straggler detection

Runtime statistics
- Worker 1: 434 images/sec
- Worker 2: 608 images/sec
- Worker 3: 592 images/sec

Autoscaling decision
- Add 2 workers
- Replace worker 1
Autoscaling Evaluation

GPU cluster: 8 machines
- 8 NVIDIA V100 GPUs (64 total)
- 64 Intel Xeon CPUs (2.2GHz)
- 250GB memory
- 16 Gbps network

CPU cluster: 60 machines
- 16 Intel Xeon CPUs @ 2.6 GHz (960 total)
- 64GB memory
- 1 Gbps network
Autoscaling Lowers JCT and GPU Time

ResNet-50 on ImageNet

ResNet-56 on CIFAR-10

45%

85.1%
Autoscaling Finds Target Configuration Quickly

ResNet-50 on ImageNet

ResNet-56 on CIFAR-10

Number of workers vs. time elapsed (s) for different configurations of workers and tasks.
Autoscaling Finds Target Configuration Quickly

<2% of total time
(train until convergence)

<6% of total time
## Autoscaling Has Short Idle Times

<table>
<thead>
<tr>
<th></th>
<th>CIFAR-10</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoscaling (+)</td>
<td>3.179</td>
<td>6.813</td>
</tr>
<tr>
<td>autoscaling (-)</td>
<td>2.612</td>
<td>4.376</td>
</tr>
<tr>
<td>checkpoint restart</td>
<td>72.756</td>
<td>81.186</td>
</tr>
</tbody>
</table>

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**
What’s missing?

Convergence guarantees; \textit{batch size} still changes...

How to preserve \textit{model semantics} across different hardware?
Resource Elasticity in Distributed Deep Learning

VirtualFlow: Decoupling Deep Learning Models from Underlying Hardware
VirtualFlow

Decoupling Deep Learning Models from the Underlying Hardware

Andrew Or, Haoyu Zhang*, Michael J. Freedman
Princeton University, *Google AI
Resource Requirements are Climbing

The task
ResNet-50
ImageNet
32768 BS

The “cluster” you have:
2 Desktop GPUs

The cluster you need
No Reproducibility

The task
ResNet-50
ImageNet
32768 BS
256 BS

You
2 Desktop GPUs

Different model convergence behavior

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
No Experimentation

The task
ResNet-50
ImageNet
32768 BS

Smaller testbed
8 V100 GPUs

Not possible today

Real deployment
Deep learning Models are tightly coupled with the underlying Hardware
Model + Hyperparameters

ResNet
BERT
Transformer

Batch size
Learning rate
Momentum

Model graph

Batch size

Physical Resources

GPU, TPU, FPGA
Batch Size Coupling

Scales with **hardware**, affects **model** convergence
Decoupling Will Enable New Use Cases

Reproducibility: Tune hyperparameters once, run everywhere

Resource elasticity: Preserve convergence across resizes

Heterogeneous training: More scheduling combinations
State-of-the-art

**Virtual nodes:** Chord, Dynamo etc.
Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications
DeCandia et al. Dynamo: Amazon’s Highly Available Key-value Store

**Experimentation:** PyTorch gradient accumulation, k-step async. training
Li et al. PyTorch Distributed: Experiences on Accelerating Data Parallel Training
Zhou et al. On the convergence properties of a K-step averaging SGD algorithm for nonconvex optimization

**Elasticity:** Cluster schedulers Gandiva, Optimus, Tiresias, Themis, Gavel
Xiao et al. Gandiva: Introspective Cluster Scheduling for Deep Learning
Gu et al. Tiresias: A GPU Cluster Manager for Distributed Deep Learning
Mahajan et al. THEMIS: Fair and Efficient GPU Cluster Scheduling
Narayanan et al. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads

**Heterogeneous Training:** Gavel, Gandiva-Fair
Narayanan et al. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads
Chaudhary et al. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning
Virtual Node Abstraction

Batch

4 servers, 16 GPUs

1 server, 4 GPUs

Processed sequentially
Virtual Node Abstraction

Flexible mapping between virtual nodes and GPUs
Virtual Node Abstraction

Flexible mapping between virtual nodes and GPUs
Virtual Node Abstraction

Flexible mapping between virtual nodes and GPUs
## Virtual Node Trade-off

<table>
<thead>
<tr>
<th>GPU requirement</th>
<th>Time req.</th>
<th>1VN per GPU</th>
<th>2VNs per GPU</th>
<th>4VNs per GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Resource req. changes while batch size stays the same
How are Virtual Nodes Executed?

**Step 1**
Forward pass + Fetch inputs

**Step 2**
Backward pass

**Step 3**
Aggregate gradients

**Step 4**
Process next VN, or sync gradients

**Step 5**
Update model

**Step 6**
Process next batch
Virtual Node Memory Overhead

**Gradient buffer:** Same size as the model

**Constant** overhead w.r.t. num virtual nodes
VirtualFlow Elasticity

**Scaling**: migrate virtual nodes across new set of resources

**Migrated state** is minimal, just another allreduce

**Batch size** is fixed during scaling

*Resource adjustments are hidden from the application*
VirtualFlow Elasticity Mechanism

Horovod allreduce function

Borrowed from (Or et al. Resource Elasticity in Distributed Deep Learning)
VirtualFlow Heterogeneous Training

Transformer on WMT14 → Offline Profiler → Heterogeneous Solver → TensorFlow / PyTorch

**Throughput**
- **K80**: 512 BS, 4 VN
- **P100**: 1024 BS, 4 VN
- **V100**: 4096 BS, 8 VN

**Correctness**: Weighted gradient synchronization, shard by local batch sizes
Heterogeneous Solver

ResNet-50 on ImageNet

Global batch size = 8192

2 V100 + 2 P100 GPUs

Even: 2048 + 2048 + 2048 + 2048

Uneven: 3072 + 3072 + 1024 + 1024

-44%
Heterogeneous Solver

Objective \quad \min_{i} \max_{i} \left( t_i(b_i) \cdot v_i + \text{comm} \right)

Constraint \quad \sum_{i} n_i \cdot b_i = B

Step time running on batch size $b_i$

Num virtual nodes

Global batch size

Communication overhead

Num GPUs
VirtualFlow Summary

Virtual nodes preserve *model semantics* across different *hardware*:

**Reproducibility:** Tune hyperparameters once, run everywhere

**Resource elasticity:** Preserve convergence across resizes

**Heterogeneous training:** More scheduling combinations
VirtualFlow Evaluation

**Environment A**
Two servers, each with:
- **8 V100 GPUs** (16 GB)
- 64 Intel Xeon CPUs (2.2GHz)
- 250 GB DRAM
- 16 Gbps network connection

**Environment B**
Two servers, each with:
- **4 P100 GPUs** (16 GB)
- 64 Intel Xeon CPUs (2.2GHz)
- 250 GB DRAM
- 16 Gbps network connection

**Environment C**
Single server:
- **2 RTX 2080Ti GPUs**
- 32 Intel(R) Xeon(R) E5-2620v4 CPUs (2.1GHz)
- 64GB DRAM
VirtualFlow Reproducibility

ResNet-50 on ImageNet for 90 epochs using a **batch size of 8192**

Previously required 32 V100 GPUs (16GB)
VirtualFlow Elasticity: 20-jobs

Jobs arrive in *poisson distribution* with an average inter-arrival time of 5 min

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Batch sizes</th>
<th>$VN_{\text{GPU}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-56</td>
<td>cifar10</td>
<td>64, 128</td>
<td>1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>ImageNet</td>
<td>256, 512, 1024</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048, 4096, 8192</td>
<td></td>
</tr>
<tr>
<td>BERT-BASE</td>
<td>CoLA</td>
<td>8, 16, 32, 64, 128</td>
<td>1, 2</td>
</tr>
<tr>
<td>BERT-BASE</td>
<td>SST-2</td>
<td>8, 16, 32, 64, 128</td>
<td>1, 2</td>
</tr>
<tr>
<td>Transformer</td>
<td>WMT</td>
<td>4096, 8192, 16384</td>
<td>1, 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32768, 65536</td>
<td></td>
</tr>
</tbody>
</table>
VirtualFlow Elasticity: 20-jobs

Cluster utilization +19.5%

-45.5%

-47.6%
VirtualFlow Heterogeneous Training

ResNet-50 on ImageNet for 90 epochs using a batch size of 8192

Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
VirtualFlow Heterogeneous Training

Extend Gavel (OSDI ‘20) to consider heterogeneous allocations

Gavel + HT
-26.4% Avg JCT
VirtualFlow decouples deep learning Models from the underlying Hardware, enabling:

**Reproducibility:** Tune hyperparameters once, run everywhere

**Resource elasticity:** +19% utilization, -47% median JCT

**Heterogeneous training:** +42% throughput, -26% average JCT
Future Directions

**Fault tolerance:** Rely on elasticity mechanisms instead of checkpoints

**Model parallelism:** Integrate with virtual nodes (more next slide)
Future Directions

Model parallelism + Data parallelism
- GPUs 6, 7
- GPUs 4, 5
- GPUs 2, 3
- GPUs 0, 1

Model parallelism + Virtual nodes
- GPU 3
- GPU 2
- GPU 1
- GPU 0
This thesis

Lower resource req.
Tune once, run everywhere
Resource elasticity
Heterogeneous training

High resource req.
Tune once per cluster
Static allocation
Homogeneous allocation
Graduate Work

**SLAQ: Quality-Driven Scheduling for Distributed Machine Learning.**

**ReLAQS: Reducing Latency for Multi-Tenant Approximate Queries via Scheduling.**
Logan Stafman, Andrew Or, Michael J. Freedman. Middleware 2019. *Best paper!*

**Resource Elasticity in Distributed Deep Learning**
Andrew Or, Haoyu Zhang, Michael J. Freedman. MLSys 2020.

**VirtualFlow: Decoupling Deep Learning Models from the Underlying Hardware**
Andrew Or, Haoyu Zhang, Michael J. Freedman. *(Under submission)*
Thank you!