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

LSDS mission statement:
“To support the design and engineering of scalable, robust and secure data-intensive applications”
Research interests and expertise

- **Systems:**
  - Distributed systems
  - Operating systems
  - Compilers
  - Networks
  - Runtime systems

- **Application domains:**
  - Data management
  - Stream processing
  - Graph processing
  - Machine learning/AI
  - Blockchain

- **Hardware & Infrastructure:**
  - Multicore CPUs
  - Trusted Hardware, TEEs
  - Accelerators/GPUs
  - Data-center networks, RDMA
  - Edge infrastructure

- **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
  [CIDE’19, SIGMOD’16]
- Stream processing
  [SIGMOD’20, EDBT’20, VLDB’17, SIGMOD’16, CIDR’15, ICDE’11]
- IoT data processing
  [VLDB’18]
- 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]

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

LSDS Group: Systems Research

Data Management

Distributed Computing

Machine Learning/AI

Security

Peter Pietzuch - Imperial College London
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**
Parameters in Distributed Deep Learning Systems

Users must tune parameters to optimise time-to-accuracy

Hyper-parameters
- Batch size
- Learning rate
- ...

System parameters
- Number of workers
- Communication topology
- ...

Small batch or large batch size?

Ring or binary-tree?
## Issues with Parameter Tuning

<table>
<thead>
<tr>
<th>Examples of empirical parameter tuning</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Change batch size at epoch 30, 60 and 90 when training with ImageNet.” [1]</td>
<td>Dataset-specific</td>
</tr>
<tr>
<td>“Linearly scale the learning rate with the #workers when training ResNet models.” [2]</td>
<td>Model-specific</td>
</tr>
<tr>
<td>“Set the topology to a ring by default.” [3]</td>
<td>Cluster-specific</td>
</tr>
</tbody>
</table>

[2] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018

Peter Pietzuch - Imperial College London
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 $\rightarrow$ keep batch size
- When GNS is large $\rightarrow$ increase batch size
Proposals for Dynamic Parameter Adaptation

Gradient variance

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

Kazuki Osawa\textsuperscript{1} Yohei Tsujii\textsuperscript{1,2} Yuichiro Ueno\textsuperscript{3} Akira Naruse\textsuperscript{3} Rio Yokota\textsuperscript{2,3} Satoshi Matsuoka\textsuperscript{4,5}

\textsuperscript{1}School of Computing, Tokyo Institute of Technology
\textsuperscript{2}Global Scientific Information and Computing Center, Tokyo Institute of Technology
\textsuperscript{3}NVIDIA
\textsuperscript{4}RIKEN Center for Computational Science

Gradient second-order metrics

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

Cost on Azure: $92,613

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

¹https://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

More efficient with larger network topologies

Better model accuracy

Adapt synchronisation strategy

More efficient with larger network topologies

Better model accuracy

HogWild!

AD-PSGD

EA-SGD

SMA

S-SGD

asynchronous

synchronous

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
   - TensorFlow
   - 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

Peter Pietzuch - Imperial College London
KungFu – Distributed Training Library

Contributions:

1. Supporting adaptation policies

2. Monitoring inside dataflow

3. Distributing parameter updates

- Monitoring, communication and adaptation functions
- Asynchronous collective communication layer
- Dynamic worker membership tables

Supports different types of adaptation
Processes large volume of monitoring data
Adapts stateful workers consistently

Supporting adaptation policies:
- GNS Policy
- Elastic Policy

Monitoring training
Adapting parameters

TensorFlow/PyTorch/Keras Workers
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:

<table>
<thead>
<tr>
<th>Monitoring</th>
<th>Communication</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• grad_noise_scale</td>
<td>• allreduce</td>
<td>• resize</td>
</tr>
<tr>
<td>• grad_variance</td>
<td>• broadcast</td>
<td>• set_tree</td>
</tr>
<tr>
<td>• ...</td>
<td>• ...</td>
<td>• ...</td>
</tr>
</tbody>
</table>

Workers

Communication

Policy

Monitoring

Adaptation
Example: Adaptation Policy for GNS

1. Adaptation logic in policy

```python
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

1. Dataflow engine launches operators asynchronously

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

Coordinating synchronous allreduce operations limits system scalability
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**

Value of # workers 10

Dataflow for averaging GNS

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

Other system parameters:
- Worker rank
- Communication topology
- …
Distributed Mechanism for Changing Parameters

**Idea:** Decouple system parameters from dataflow state

1. System parameters as computational operators
   - gns \(\rightarrow\) allreduce \(\rightarrow\) avg \(\rightarrow\) size_op
   - Dynamic worker membership
   - KungFu communication layer

2. Update worker membership using collective communication
   - Parameter update
   - Membership

Always obtains up-to-date view of system parameters
**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?

GNS policy, CIFAR-10 ResNet, 4 GPUs

Small batch size reaches high accuracy, but converges slowly
How Effectively Does KungFu Adapt?

Large batch size finishes quickly, but accuracy suffers

GNS policy, CIFAR-10 ResNet, 4 GPUs

Large batch size
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**

Switching from S-SGD to AD-PSGD
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

Thank You — Any Questions?