
AWS
re:Invent

AIM402

Fast distributed training and near linear scaling with distributed DataParallel in Amazon SageMaker

Aditya Bindal

Sr Product Manager, AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Why use data parallelism

How does data parallelism work

What is hard about data parallelism?

SageMaker Distributed DataParallel: Main features

Using data parallelism in SageMaker with PyTorch and TensorFlow



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why use data parallelism



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Long training times slow down development



Credit: https://xkcd.com/303/



Model	RoBERTa
Dataset	300+ GB
Cluster	64 p3dn.24xl
Training time	4 days

Scaling to more GPUs can reduce training time



1 Node
Time-to-train: T hours
Cost: T * price

8 Nodes
Time-to-train: (T/8)/P% hours
Cost: (T/8)/P% * price

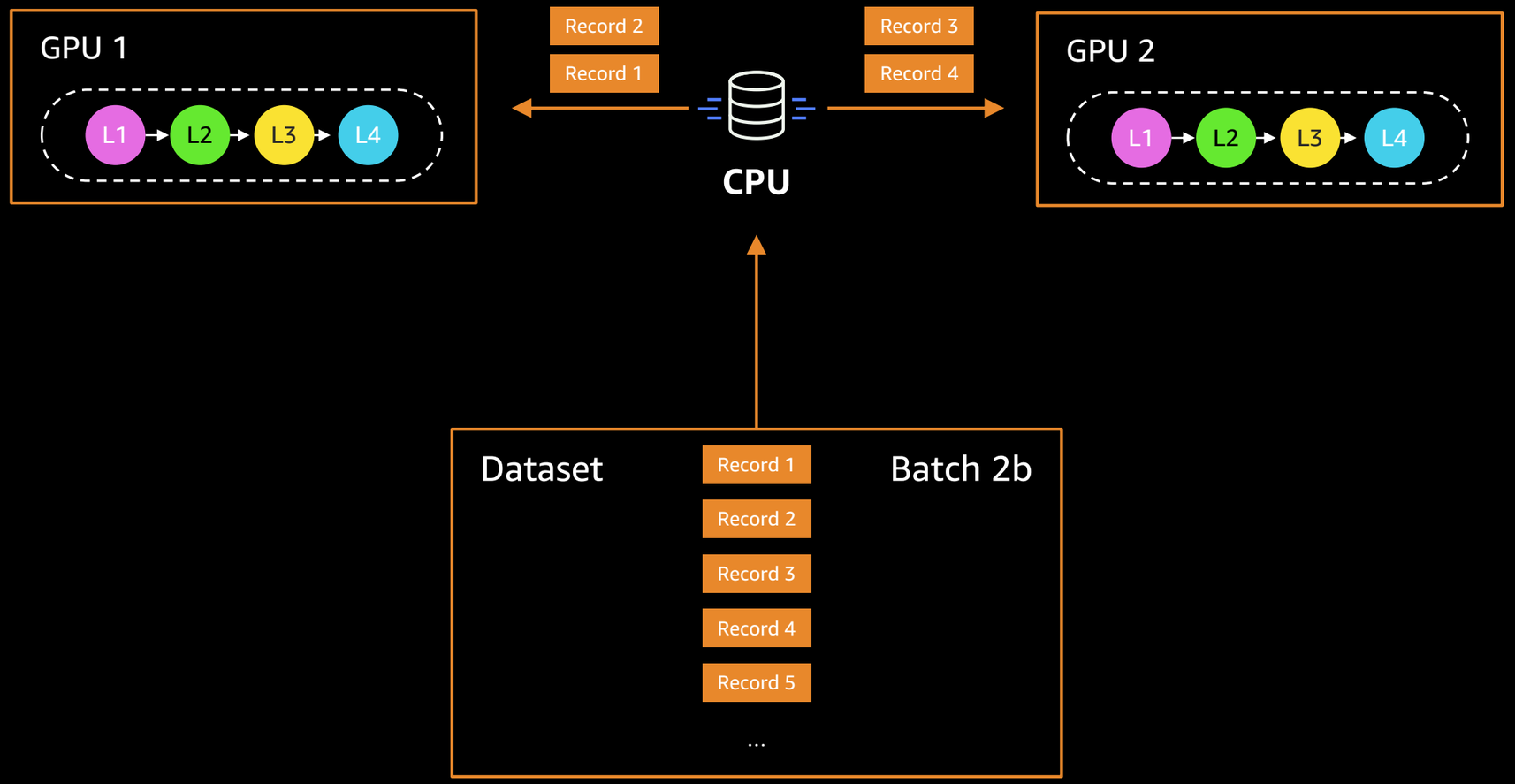
Example: With 90% scaling efficiency from 1 to 8 nodes,
training time decreases by 86% and cost increases by 11%

How does data parallelism work

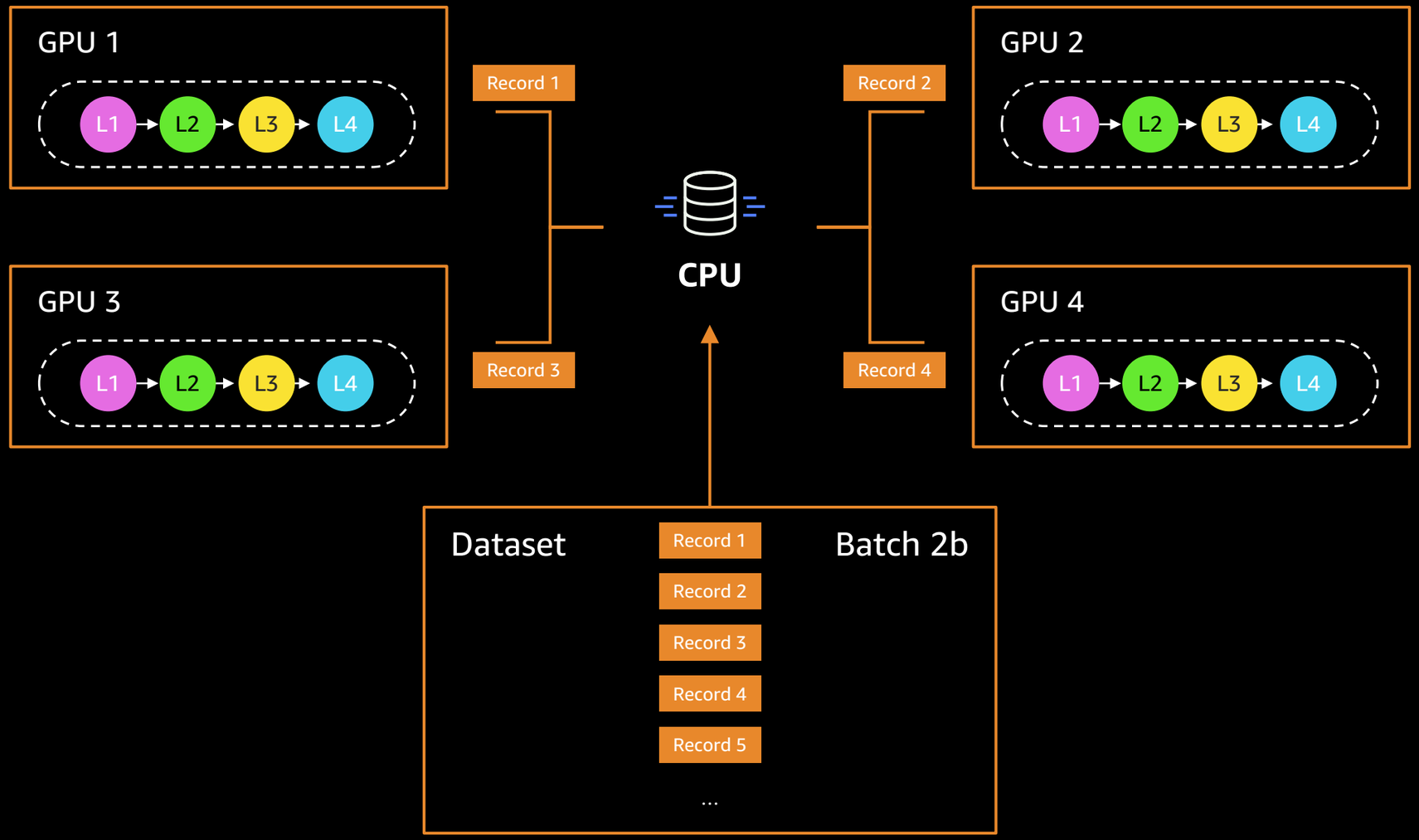


© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

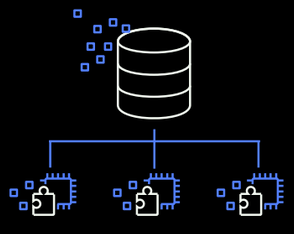
Quick overview of data parallelism



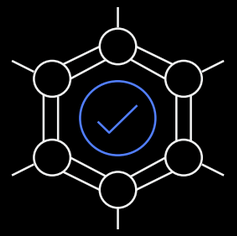
Quick overview of data parallelism



Gradient synchronization



Parameter server
E.g., TensorFlow
ParameterServerStrategy



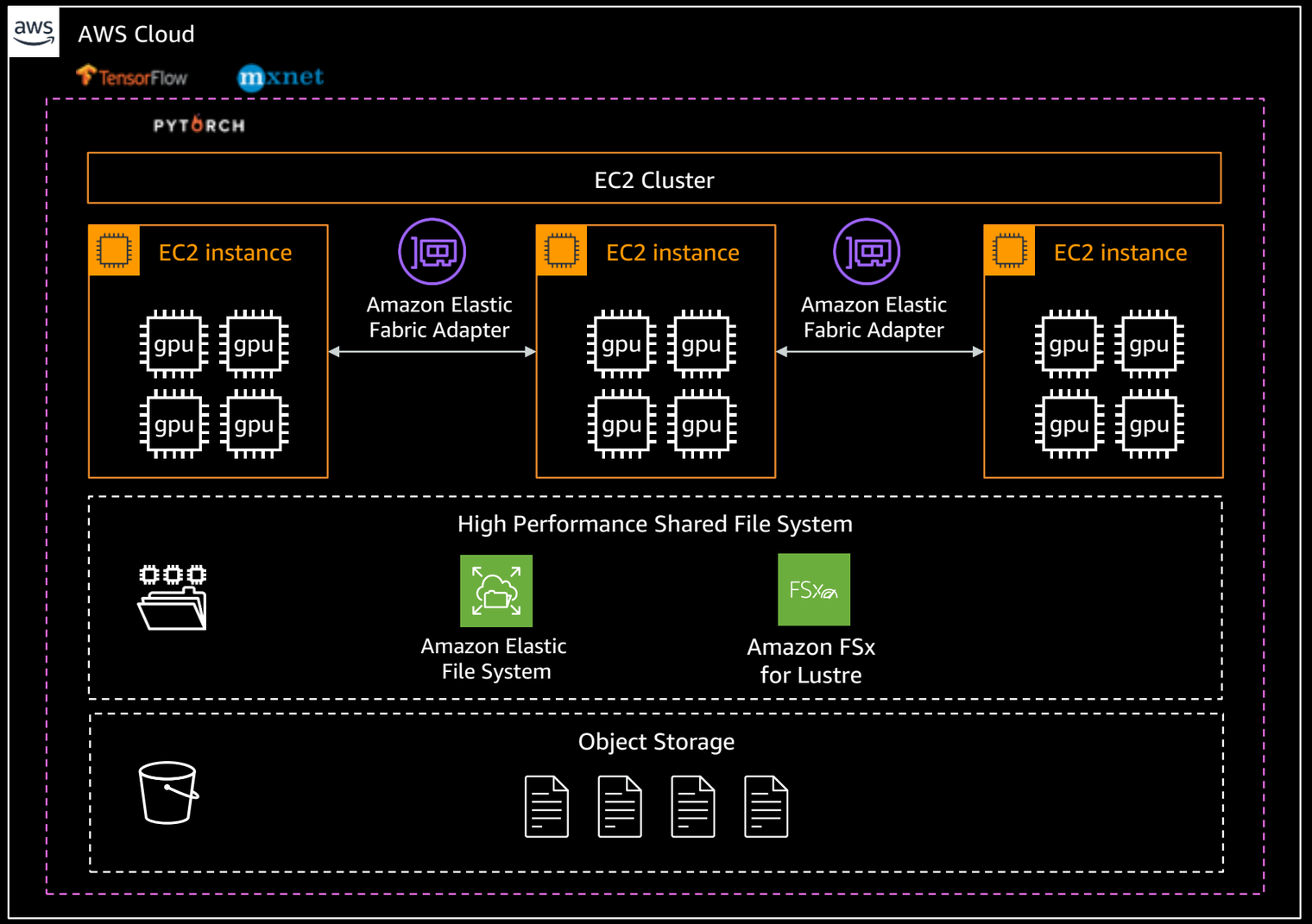
MPI AllReduce
E.g., Horovod, PyTorch
DistributedDataParallel

What is hard about data parallelism?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Infrastructure

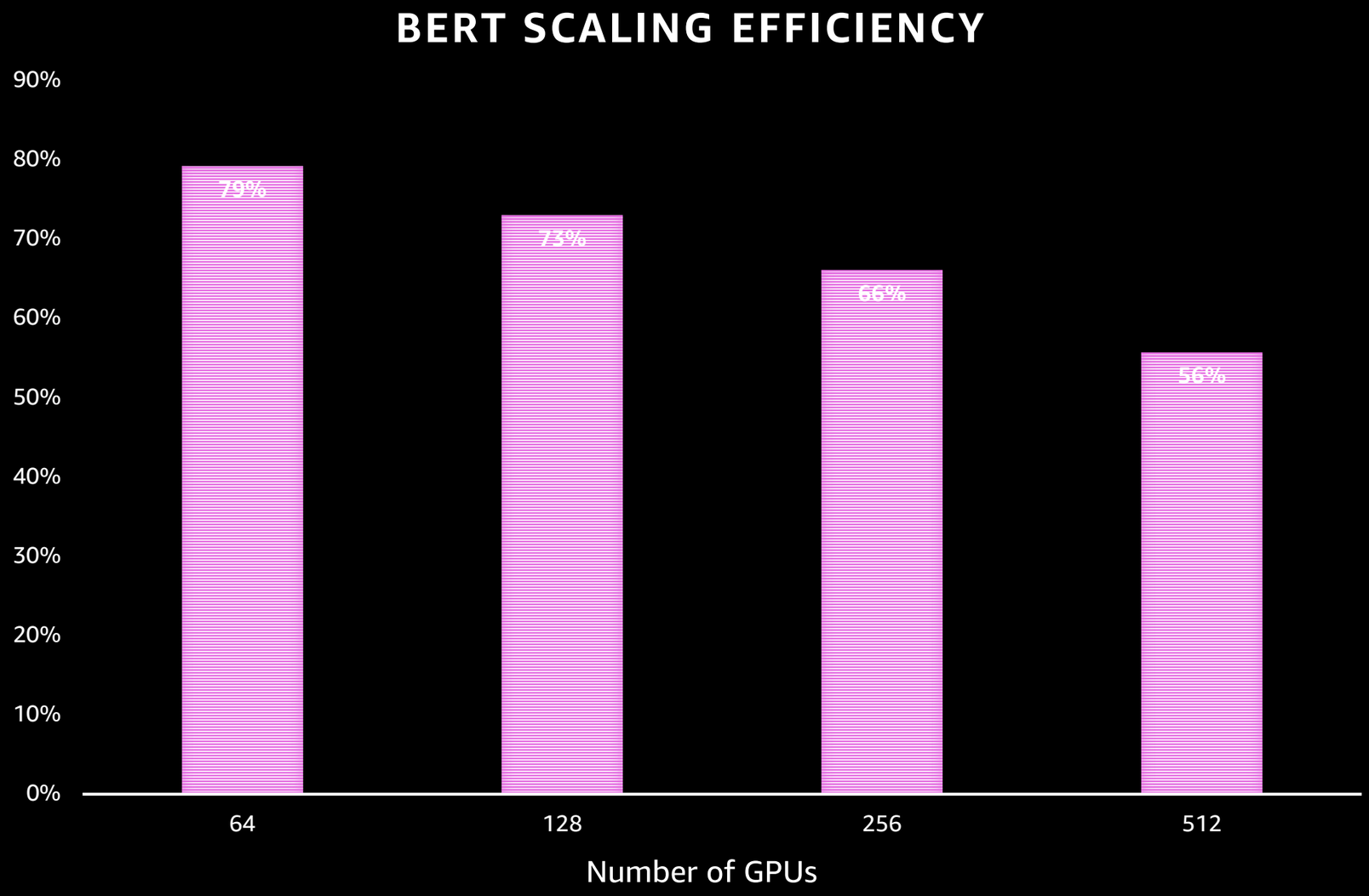


TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Performance



How other solutions scale



BERT-large using NCCL on p3dn.24xlarge clusters

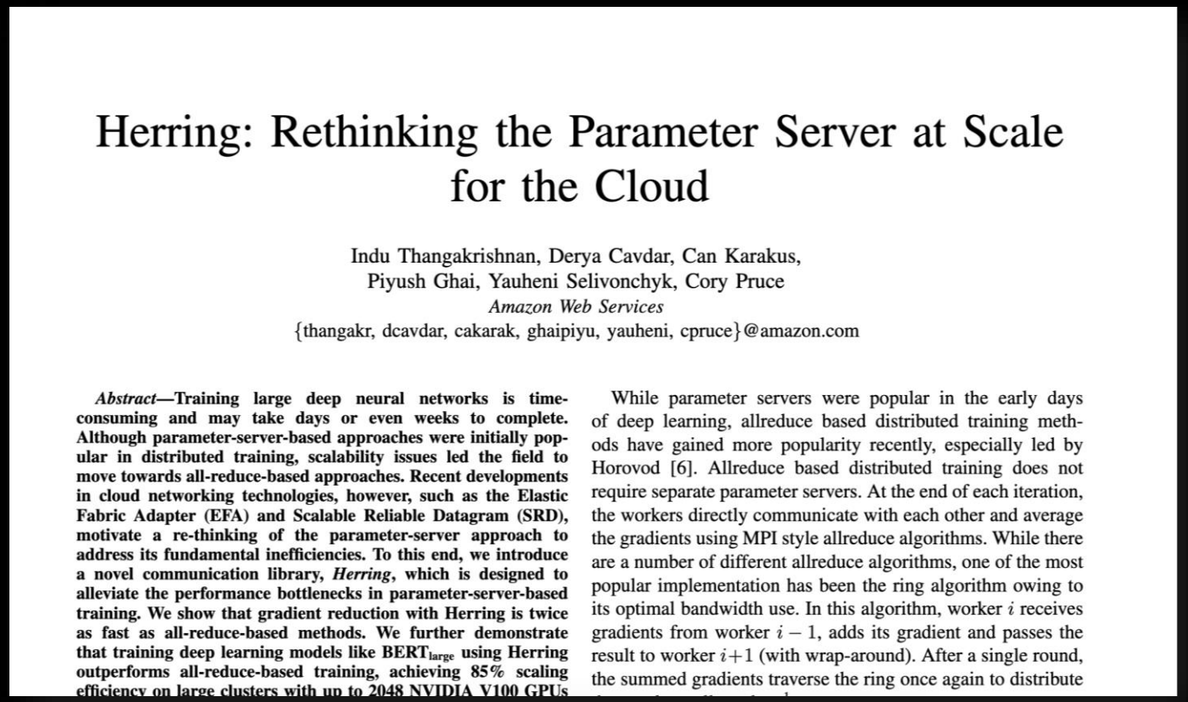
SageMaker Distributed DataParallel: Main features



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DataParallel in SageMaker Distributed



Herring: Rethinking the Parameter Server at Scale for the Cloud

Indu Thangakrishnan, Derya Cavdar, Can Karakus,
Piyush Ghai, Yauheni Selivonchyk, Cory Pruce
Amazon Web Services
{thangakr, dcavdar, cakarak, ghaipiyu, yauheni, cpruce}@amazon.com

***Abstract*—Training large deep neural networks is time-consuming and may take days or even weeks to complete. Although parameter-server-based approaches were initially popular in distributed training, scalability issues led the field to move towards all-reduce-based approaches. Recent developments in cloud networking technologies, however, such as the Elastic Fabric Adapter (EFA) and Scalable Reliable Datagram (SRD), motivate a re-thinking of the parameter-server approach to address its fundamental inefficiencies. To this end, we introduce a novel communication library, *Herring*, which is designed to alleviate the performance bottlenecks in parameter-server-based training. We show that gradient reduction with Herring is twice as fast as all-reduce-based methods. We further demonstrate that training deep learning models like $\text{BERT}_{\text{large}}$ using Herring outperforms all-reduce-based training, achieving 85% scaling efficiency on large clusters with up to 2048 NVIDIA V100 GPUs**

While parameter servers were popular in the early days of deep learning, allreduce based distributed training methods have gained more popularity recently, especially led by Horovod [6]. Allreduce based distributed training does not require separate parameter servers. At the end of each iteration, the workers directly communicate with each other and average the gradients using MPI style allreduce algorithms. While there are a number of different allreduce algorithms, one of the most popular implementation has been the ring algorithm owing to its optimal bandwidth use. In this algorithm, worker i receives gradients from worker $i-1$, adds its gradient and passes the result to worker $i+1$ (with wrap-around). After a single round, the summed gradients traverse the ring once again to distribute

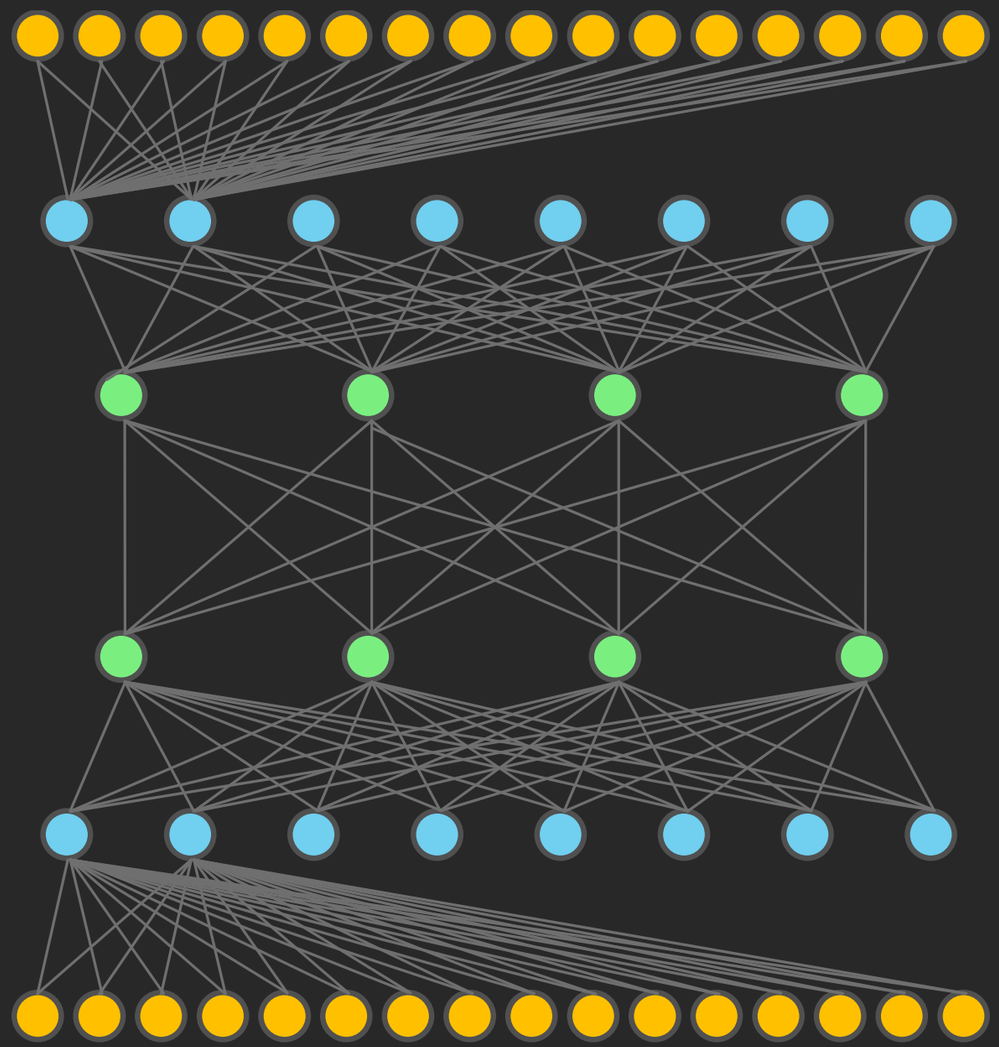
2018	Fastest Resnet
2019	Fastest BERT and Mask-RCNN
2020	Fastest T5-3B and Mask-RCNN

Presented at SC 20

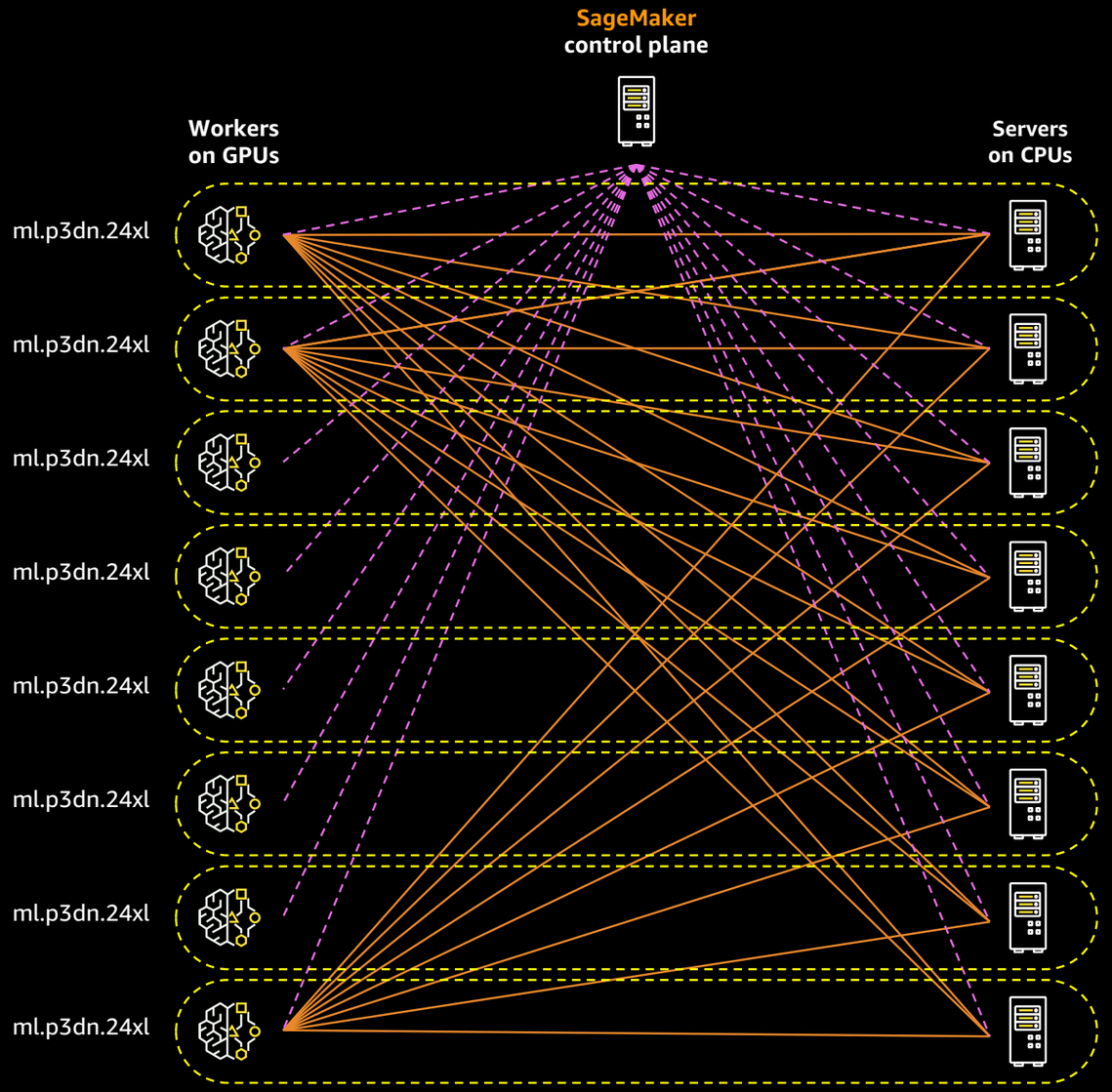
https://www.amazon.science/publications/herring-rethinking-the-parameter-server-at-scale-for-the-cloud

DataParallel in SageMaker Distributed

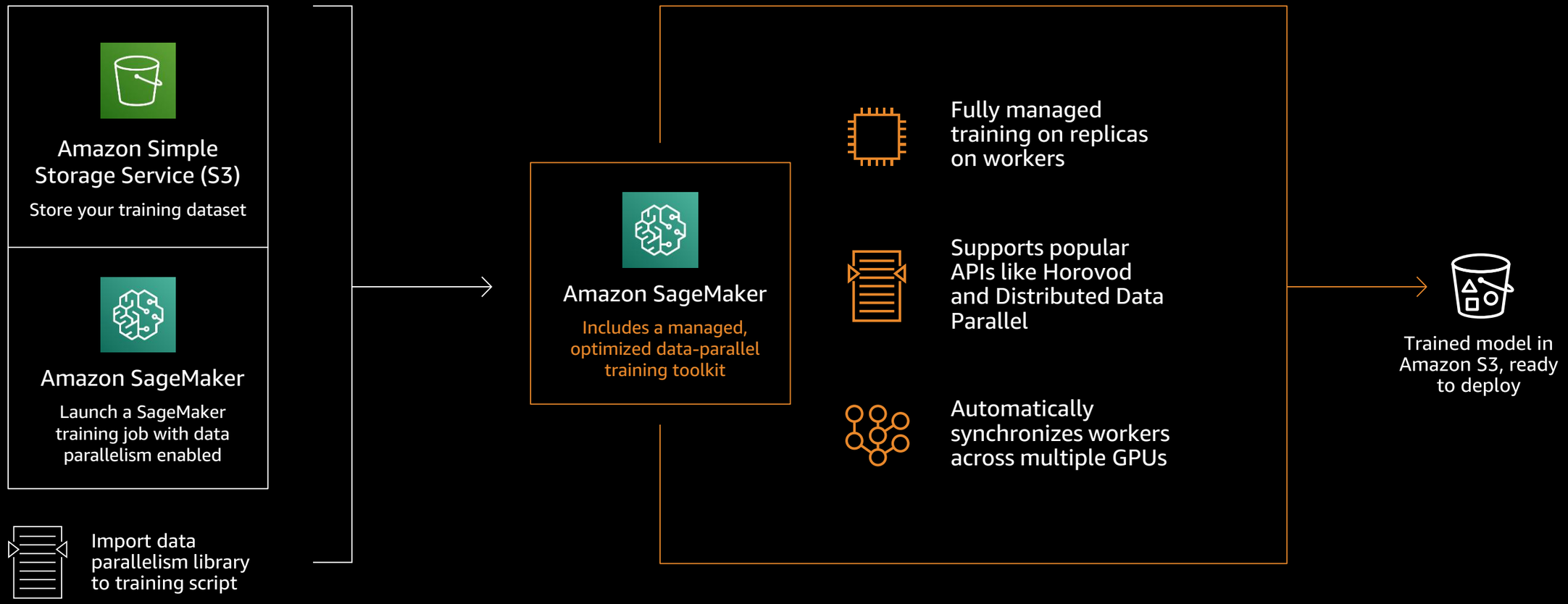
- Library for distributed training of deep learning models in TensorFlow, PyTorch
- Accelerates training for network-bound workloads
- Built and optimized for AWS network topology and hardware
- **20%–40% faster and cheaper than NCCL and MPI-based solutions. Best performance on AWS**



DataParallel in SageMaker Distributed



DataParallel in SageMaker Distributed



DataParallel in SageMaker Distributed

Model	Throughput			Scaling Efficiency		
	PT-DDP (samples/sec)	SageMaker DataParallel (samples/sec)	Improvement	PT-DDP	SageMaker DataParallel	Improvement
Mask R-CNN 4 node p3dn.24xl	258	307	19%	70%	83%	13%
Mask R-CNN 8 node p3dn.24xl	545	617	13%	74%	84%	10%
BERT-Large 4 node p3dn.24xl	3017	4603	52%	55%	84%	29%
BERT-Large 8 node p3dn.24xl	7409	8551	15%	67%	78%	11%

DataParallel in SageMaker Distributed

TensorFlow

- import horovod.tensorflow as hvd → *import smdistributed.dataparallel.tensorflow as dist*
- And, replace all hvd.API(...) *to* dist.API(...);
- Example: hvd.allreduce(..) → dist.allreduce(..)

PyTorch

- import torch.distributed as dist → import smdistributed.dataparallel.torch as dist
- import nn.parallel.DistributedDataParallel as DDP →

  from smdistributed.dataparallel.torch import DistributedDataParallel as DDP

Using data parallelism in SageMaker with PyTorch



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Using data parallelism with PyTorch

STEP 1: ADD dataparallel ARGUMENT IN SAGEMAKER ESTIMATOR

```
estimator = PyTorch(base_job_name='pytorch-smdataparallel-mnist',
                    source_dir='code',
                    entry_point='train_pytorch_smdataparallel_mnist.py',
                    role=role,
                    framework_version='1.6.0',
                    instance_count=2,
                    instance_type= 'ml.p3.16xlarge',
                    sagemaker_session=sagemaker_session,
                    distribution={'smdistributed':{
                                    'dataparallel':{
                                            'enabled': True
                                        }
                                    }
                                },
                    debugger_hook_config=False)
```

Enable SM dataparallel in the PyTorch Estimator in the SageMaker Python SDK

Using data parallelism with PyTorch

STEP 2: MODIFY TRAINING SCRIPT

```
estimator = PyTorch(base_job_name='pytorch-smdataparallel-mnist',
                    source_dir='code',
                    entry_point='train_pytorch_smdataparallel_mnist.py',
                    role=role,
                    framework_version='1.6.0',
                    instance_count=2,
                    instance_type= 'ml.p3.16xlarge',
                    sagemaker_session=sagemaker_session,
                    distribution={'smdistributed':{
                                      'dataparallel':{
                                              'enabled': True
                                          }
                                  }
                                },
                    debugger_hook_config=False)
```

Using data parallelism with PyTorch

STEP 2: IMPORT AND INITIALIZE SAGEMAKER dataparallel LIBRARY

```
# SDP: Import SDP PyTorch API
import smdistributed.dataparallel.torch.distributed as dist

# SDP: Import SDP PyTorch DDP
from smdistributed.dataparallel.torch.parallel.distributed import DistributedDataParallel as DDP

# SDP: Initialize SDP
dist.init_process_group()
```

Using data parallelism with PyTorch

NO CHANGE TO CORE TRAINING LOOP

```
class Net(nn.Module):
    ...
    # Define model

def train(...):
    ...
    # Model training

def test(...):
    ...
    # Model evaluation
```

No changes to model definition, core training loop, evaluation.

Using data parallelism with PyTorch

STEP 3: REPLACE PYTORCH DDP WITH SAGEMAKER DISTRIBUTED DATAPARALLEL (SDP)

```
def main():

    # SDP: Scale batch size by world size
1   batch_size //= dist.get_world_size() // 8
    batch_size = max(batch_size, 1)

    # Prepare dataset
    train_dataset = torchvision.datasets.MNIST(...)

    # SDP: Set num_replicas and rank in DistributedSampler
    train_sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,
2       num_replicas=dist.get_world_size(),
        rank=dist.get_rank())

    train_loader = torch.utils.data.DataLoader(..)

    # SDP: Wrap the PyTorch model with SDP's DDP
3   model = DDP(Net().to(device))

    # SDP: Pin each GPU to a single SDP process.
    torch.cuda.set_device(local_rank)
4   model.cuda(local_rank)

    # Train
    optimizer = optim.Adadelta(...)
    scheduler = StepLR(...)
    for epoch in range(1, args.epochs + 1):
        train(...)
        if rank == 0:
            test(...)
        scheduler.step()

    # SDP: Save model on master node.
5   if dist.get_rank() == 0:
        torch.save(...)

if __name__ == '__main__':
    main()
```

1. Scale batch size by number of processes
2. Set `number_replicas` = number of GPUs in the cluster. Set the node rank in the distributed sampler
3. Wrap the model with `DDP(model)`
4. Pin each GPU to a single SDP process
5. Save checkpoints on the leader node

Using data parallelism in SageMaker with TensorFlow



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Using data parallelism with TensorFlow

STEP 1: ADD dataparallel ARGUMENT IN SAGEMAKER ESTIMATOR API

```
estimator = TensorFlow(entry_point='albert/run_pretraining.py',
                       role=role,
                       image_uri=docker_image,
                       source_dir='deep-learning-models/models/nlp',
                       framework_version='2.3.1',
                       train_instance_count=instance_count,
                       train_instance_type=instance_type,
                       sagemaker_session=sagemaker_session,
                       subnets=subnets,
                       hyperparameters=hyperparameters,
                       security_group_ids=security_group_ids,
                       debugger_hook_config=False,
                       distribution={'smdistributed':{
                                        'dataparallel':{
                                                'enabled': True
                                            }
                                        }
                                    }
                       )
```

Enable SM dataparallel in the TensorFlow Estimator in the SageMaker Python SDK

Using data parallelism with TensorFlow

STEP 2: MODIFY TRAINING SCRIPT

```
estimator = TensorFlow(entry_point='albert/run_pretraining.py',
                       role=role,
                       image_uri=docker_image,
                       source_dir='deep-learning-models/models/nlp',
                       framework_version='2.3.1',
                       train_instance_count=instance_count,
                       train_instance_type=instance_type,
                       sagemaker_session=sagemaker_session,
                       subnets=subnets,
                       hyperparameters=hyperparameters,
                       security_group_ids=security_group_ids,
                       debugger_hook_config=False,
                       distribution={'smdistributed':{
                                        'dataparallel':{
                                                'enabled': True
                                            }
                                        }
                                    }
                      )
```

Using data parallelism with TensorFlow

STEP 2: IMPORT AND INITIALIZE SAGEMAKER dataparallel LIBRARY

```
import tensorflow as tf

# SDP: Import SDP TF API
import smdistributed.dataparallel.tensorflow as sdp

# SDP: Initialize SDP
sdp.init()
```

Using data parallelism with TensorFlow

NO CHANGE TO CORE TRAINING LOOP

```
# Prepare Dataset
dataset = tf.data.Dataset.from_tensor_slices(...)

# Define Model
mnist_model = tf.keras.Sequential(...)
loss = tf.losses.SparseCategoricalCrossentropy()

# SDP: Scale Learning Rate
# LR for 8 node run : 0.000125
# LR for single node run : 0.001
opt = tf.optimizers.Adam(0.000125 * sdp.size())
```

No changes to model definition, core training loop, evaluation.

Scale learning rate by the number of workers, using `sdp.size()`

Using data parallelism with TensorFlow

STEP 3: REPLACE TENSORFLOW HOROVOD WITH SAGEMAKER DISTRIBUTED DATAPARALLEL (SDP)

```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    # SDP: Pin GPUs to a single SDP process
1   tf.config.experimental.set_visible_devices(gpus[sdp.local_rank()], 'GPU')
```

```
@tf.function
def training_step(images, labels, first_batch):
    with tf.GradientTape() as tape:
        probs = mnist_model(images, training=True)
        loss_value = loss(labels, probs)

    # SDP: Wrap tf.GradientTape with SDP's DistributedGradientTape
2   tape = sdp.DistributedGradientTape(tape)

    grads = tape.gradient(loss_value, mnist_model.trainable_variables)
    opt.apply_gradients(zip(grads, mnist_model.trainable_variables))

    if first_batch:
       # SDP: Broadcast model and optimizer variables
3      sdp.broadcast_variables(mnist_model.variables, root_rank=0)
       sdp.broadcast_variables(opt.variables(), root_rank=0)

    return loss_value

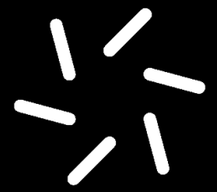
# SDP: Save checkpoints only from master node.
4 if sdp.rank() == 0:
    checkpoint.save(checkpoint_dir)
```

1. Pin each GPU to a SDP process with local rank
2. Wrap `tf.GradientTape` with `sdp.DistributedGradientTape`
3. Broadcast model variables from leader node to all other worker nodes
4. Save checkpoints on the leader node

“Training of our simulation is something we continuously iterate on, but on a single machine each training takes days, hindering our ability to iterate on new ideas quickly. We are very excited about distributed training on Amazon SageMaker, which we are expecting to decrease our training times by 90% and to help us focus on our main task: to write a best-of-the-breed codebase for the cell model training.”

Kristóf Szalay

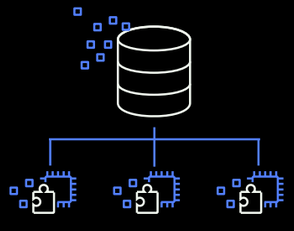
Chief Technology Officer



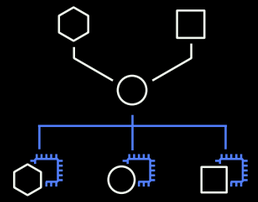
turbine™

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

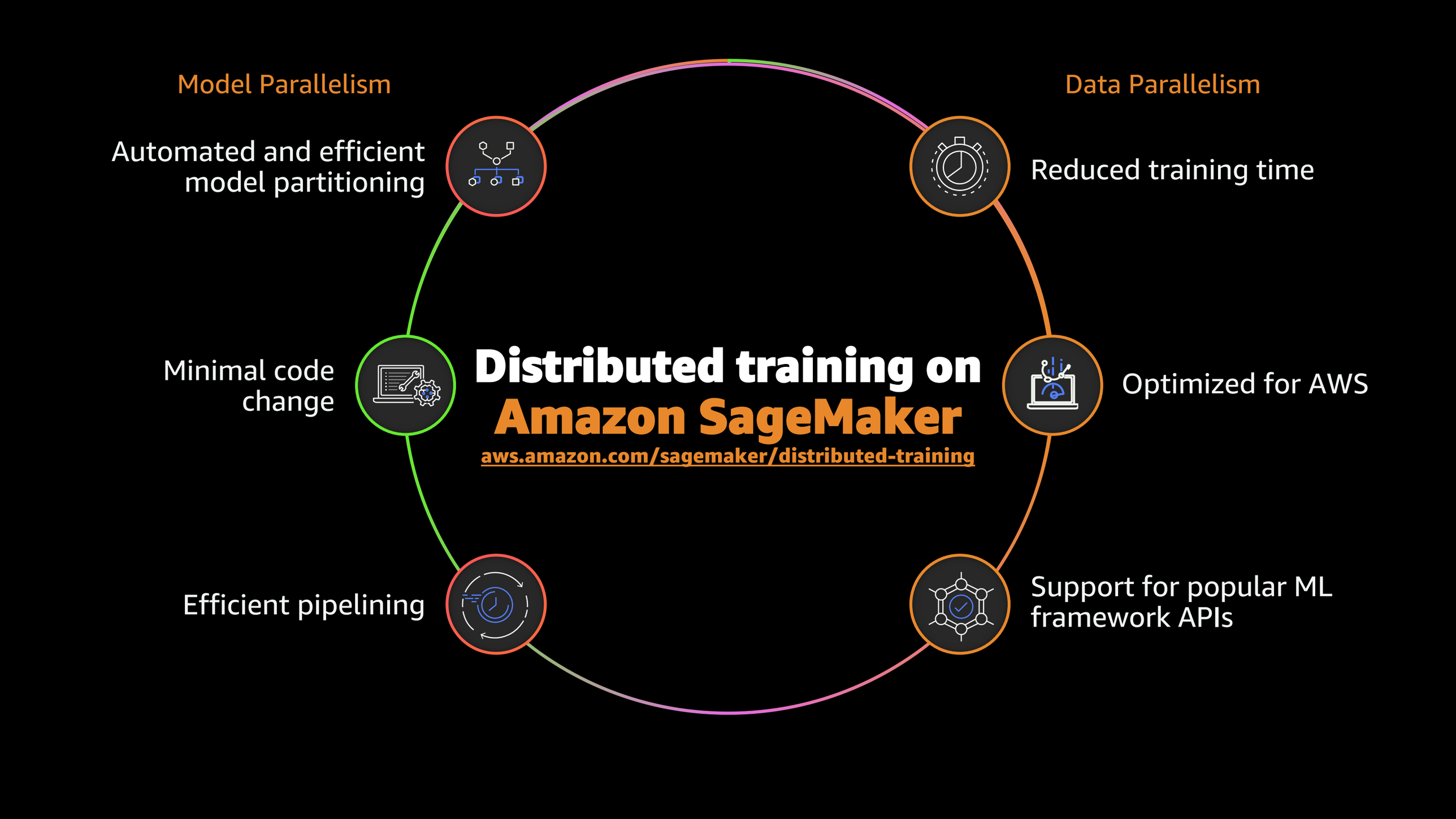
Distributed training on SageMaker



Data Parallelism
This session (AIM402)



Model Parallelism
AIM405


Model Parallelism
Automated and efficient model partitioning
Minimal code change
Efficient pipelining
Data Parallelism
Reduced training time
Optimized for AWS
Support for popular ML framework APIs
Distributed training on Amazon SageMaker
aws.amazon.com/sagemaker/distributed-training

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.