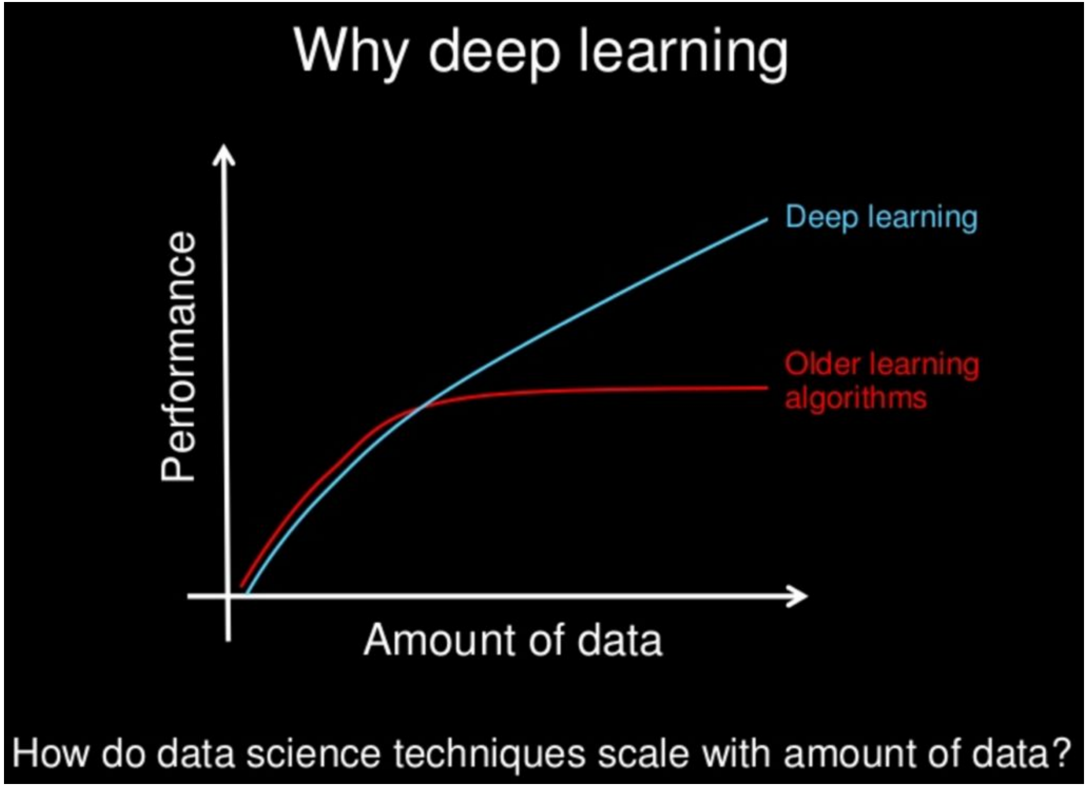
Uber's Distributed Deep Learning Journey

Alex Sergeev, Machine Learning Platform, Uber Engineering
@alsrgv

UBER

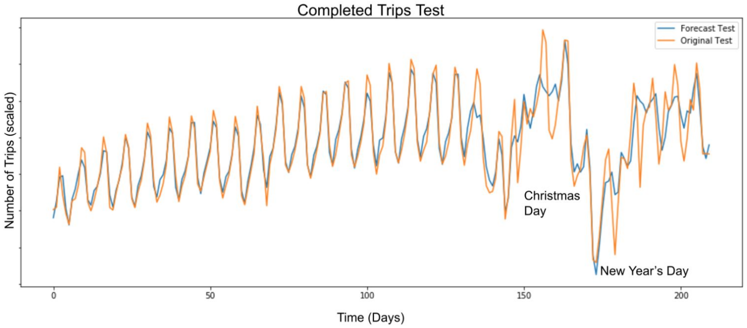
Deep Learning



Credit: Andrew Ng, https://www.slideshare.net/ExtractConf

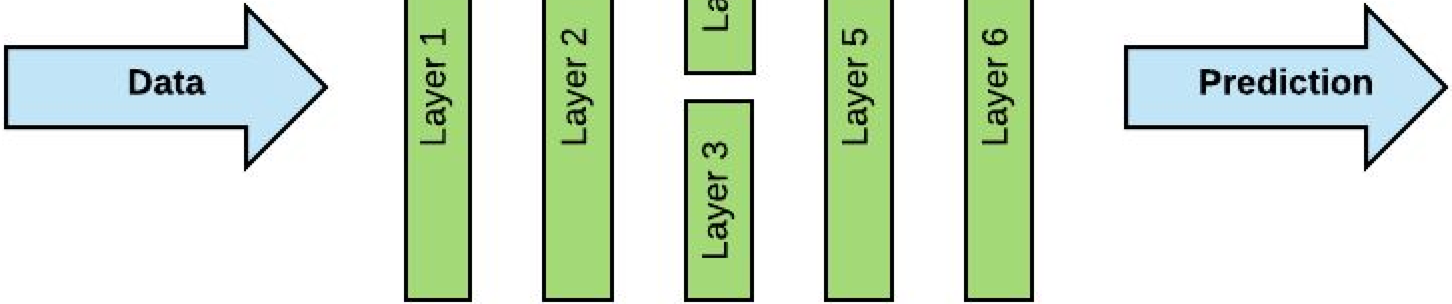
Deep Learning @ Uber

- Self-Driving Vehicles
- Trip Forecasting
- Fraud Detection
- ... and much more!

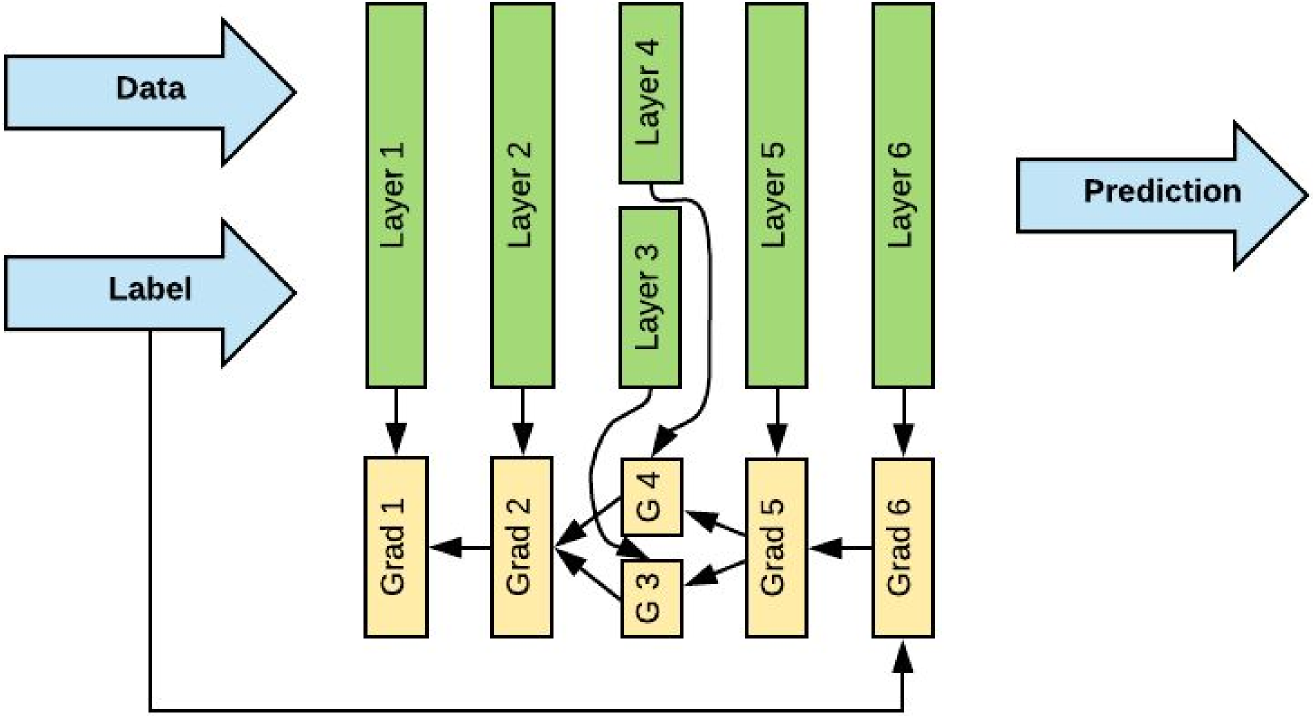


UBER

How does Deep Learning work?



How does Deep Learning training work?

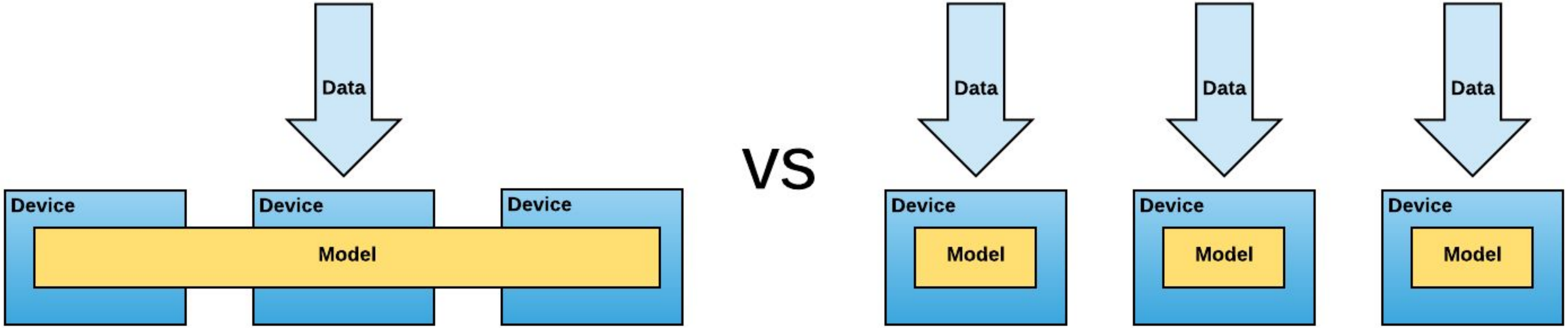


TensorFlow

- Most popular open source framework for deep learning
- Combines high performance with ability to tinker with low level model details
- Has end-to-end support from research to production

Going Distributed

- Train very large models
- Speed up model training

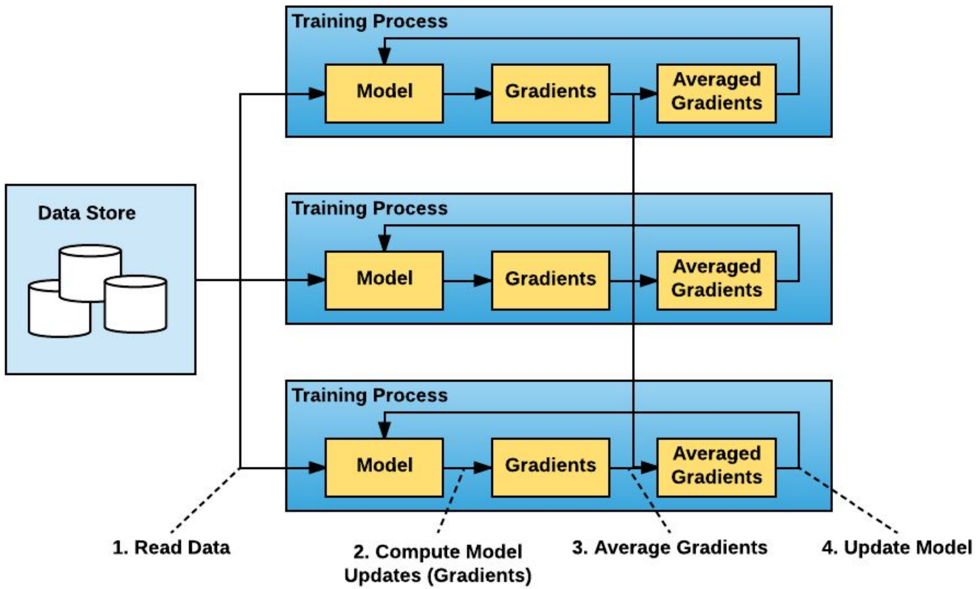


Model Parallelism

Data Parallelism

Going Distributed Cont.

- Modern GPUs have a lot of RAM
- Vast majority of use cases are data-parallel
- Facebook demonstrated training ResNet-50 on ImageNet in 1 hour (arxiv.org/abs/1706.02677)



Parameter Server Technique

tf.Server()

tf.train.replicas_device_setter()

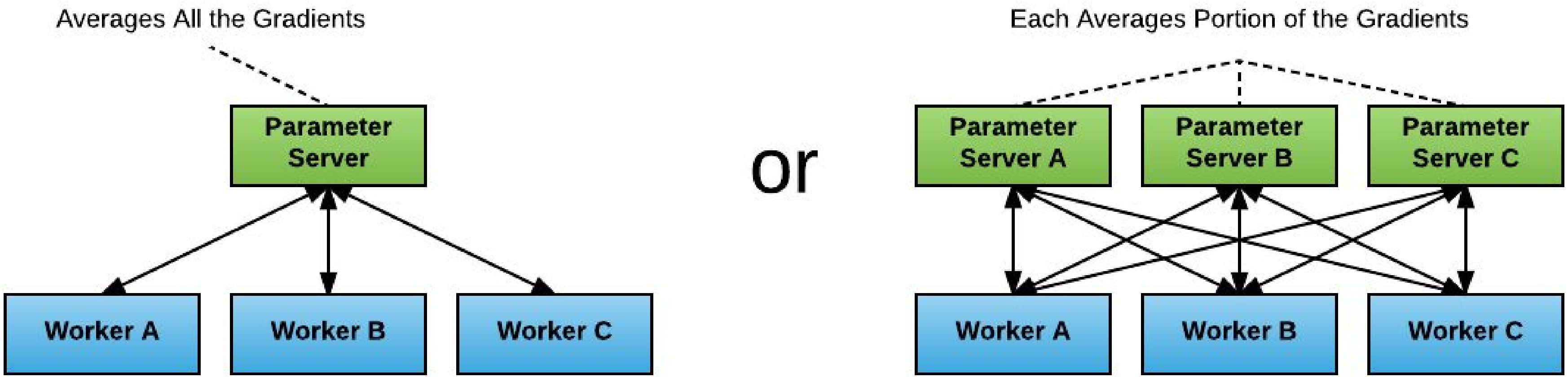
Parameter Server

Worker

GPU Towers

tf.ClusterSpec()

tf.train.SyncReplicasOptimizer()



Parameter Server Technique - Example Script

```python
import argparse
import sys

import tensorflow as tf

FLAGS = None

def main(_):
  ps_hosts = FLAGS.ps_hosts.split(",")
  worker_hosts = FLAGS.worker_hosts.split(",")

  # Create a cluster from the parameter server and worker hosts.
  cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

  # Create and start a server for the local task.
  server = tf.train.Server(cluster,
                           job_name=FLAGS.job_name,
                           task_index=FLAGS.task_index)

  if FLAGS.job_name == "ps":
    server.join()
  elif FLAGS.job_name == "worker":

    # Assigns ops to the local worker by default.
    with tf.device(tf.train.replica_device_setter(
        worker_device="/job:worker/task:%d" % FLAGS.task_index,
        cluster=cluster)):

      # Build model...
      loss = ...
      global_step = tf.contrib.framework.get_or_create_global_step()

      train_op = tf.train.AdagradOptimizer(0.01).minimize(
          loss, global_step=global_step)

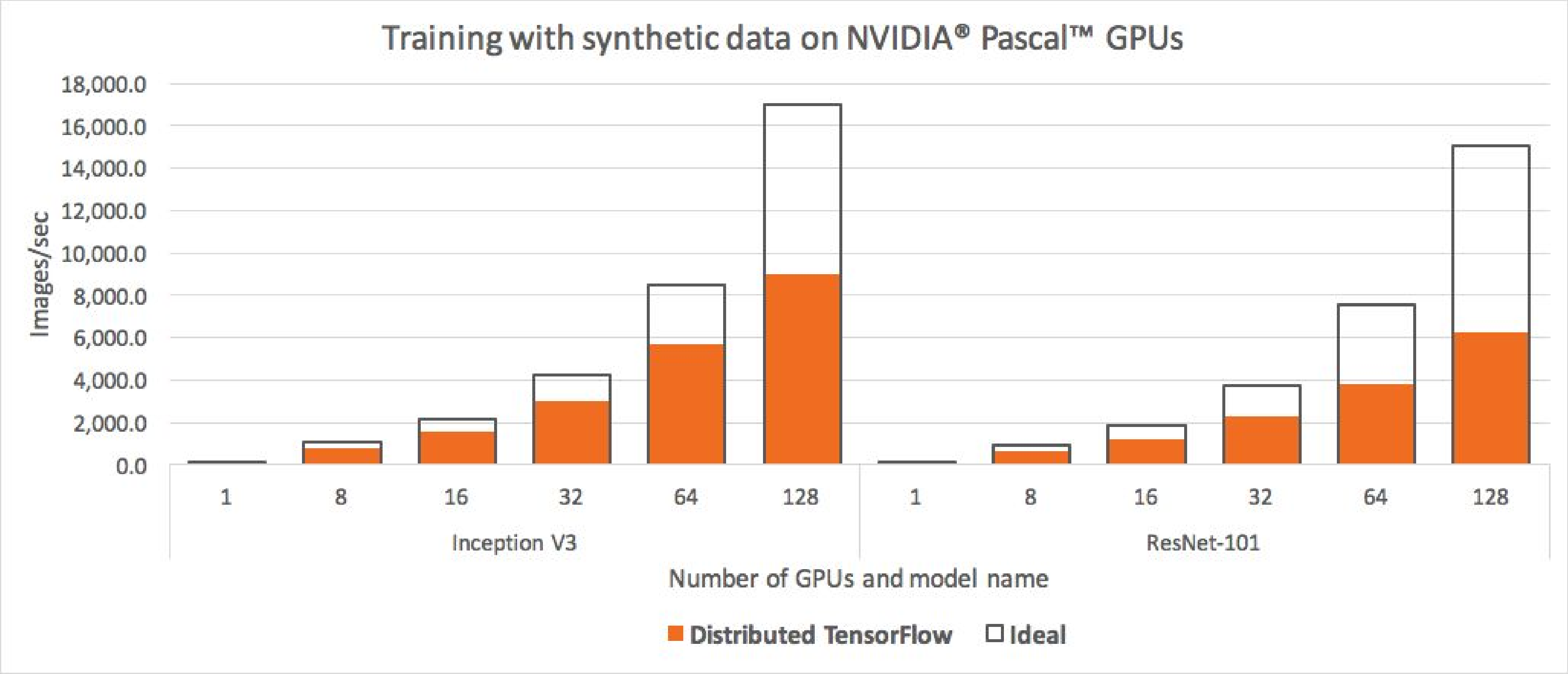
    # The StopAtStepHook handles stopping after running given steps.
    hooks=[tf.train.StopAtStepHook(last_step=1000000)]

    # The MonitoredTrainingSession takes care of session initialization,
    # restoring from a checkpoint, saving to a checkpoint, and closing when done
    # or an error occurs.
    with tf.train.MonitoredTrainingSession(master=server.target,
                                           is_chief=(FLAGS.task_index == 0),
                                           checkpoint_dir="/tmp/train_logs",
                                           hooks=hooks) as mon_sess:
      while not mon_sess.should_stop():
        # Run a training step asynchronously.
        # See `tf.train.SyncReplicasOptimizer` for additional details on how to
        # perform *synchronous* training.
        # mon_sess.run handles AbortedError in case of preempted PS.
        mon_sess.run(train_op)

if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.register("type", "bool", lambda v: v.lower() == "true")
  # Flags for defining the tf.train.ClusterSpec
  parser.add_argument(
      "--ps_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--worker_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--job_name",
      type=str,
      default="",
      help="One of 'ps', 'worker'"
  )
  # Flags for defining the tf.train.Server
  parser.add_argument(
      "--task_index",
      type=int,
      default=0,
      help="Index of task within the job"
  )
  FLAGS, unparsed = parser.parse_known_args()
```

Image Source: TensorFlow -- https://www.tensorflow.org/deploy/distributed

Parameter Server Technique - Performance



UBER

Considering ImageNet dataset of 1.3M images, this allows to train ResNet-101 for one epoch in 3.5 minutes. Scaling efficiency on 128 GPUs is only 42%, however.

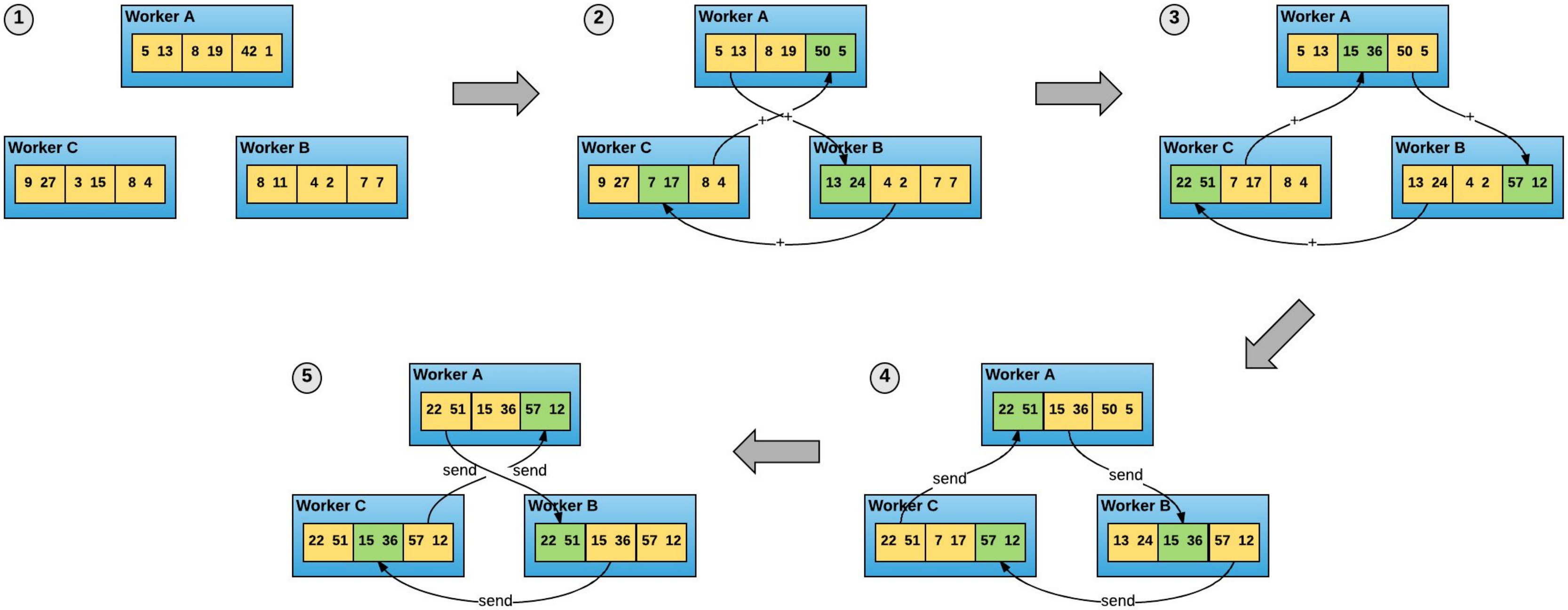
How Can We Improve?

- Re-think necessary complexity for data-parallel case
- Improve communication algorithm
- Use RDMA-capable networking (InfiniBand, RoCE)

Meet Horovod

- Distributed training framework for TensorFlow
- Inspired by HPC techniques and work of Baidu, Facebook, et al.
- Uses bandwidth-optimal communication protocols
  - Makes use of RDMA (InfiniBand, RoCE) if available
- Seamlessly installs on top of TensorFlow via `pip install horovod`
- Named after traditional Russian folk dance where participants dance in a circle with linked hands

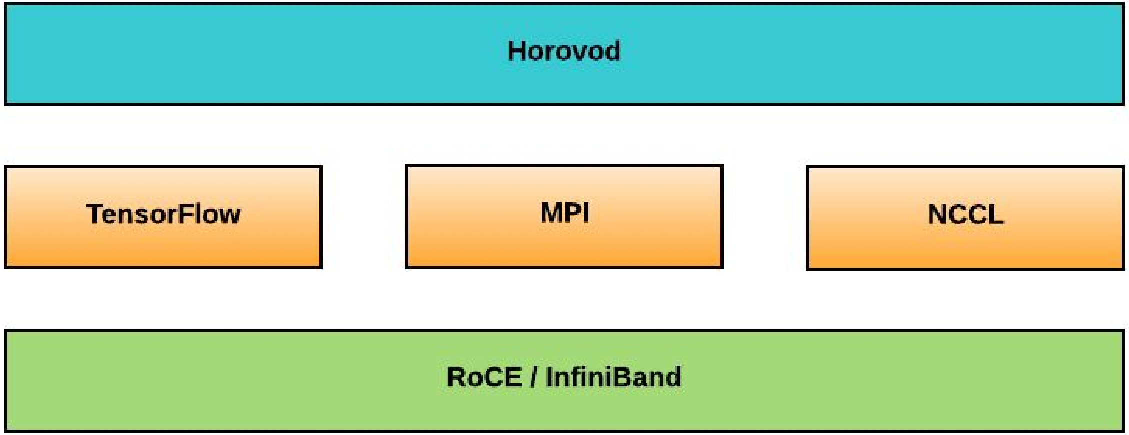
Horovod Technique



Patarasuk, P., & Yuan, X. (2009). Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2), 117-124. doi:10.1016/j.jpdc.2008.09.002

Horovod Stack

- Plugs into TensorFlow via custom op mechanism
- Uses MPI for worker discovery and reduction coordination
- Uses NVIDIA NCCL for actual reduction on the server and across servers



Horovod Example

```
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                       config=config, hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

UBER

Horovod Example - Keras

```
import keras
from keras import backend as K
import tensorflow as tf
import horovod.keras as hvd

# Initialize Horovod.
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))

# Build model…
model = ...
opt = keras.optimizers.Adadelta(1.0)

# Add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

model.compile(loss=keras.losses.categorical_crossentropy, optimizer=opt, metrics=['accuracy'])

# Broadcast initial variable states from rank 0 to all other processes.
callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]

model.fit(x_train, y_train,
        callbacks=callbacks,
        epochs=10,
        validation_data=(x_test, y_test))
```

Horovod Example - Estimator API

```
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
def model_fn(features, labels, mode):
  loss = …
  opt = tf.train.AdagradOptimizer(0.01)

  # Add Horovod Distributed Optimizer
  opt = hvd.DistributedOptimizer(opt)

  train_op = optimizer.minimize(loss=loss, global_step=tf.train.get_global_step())
  return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

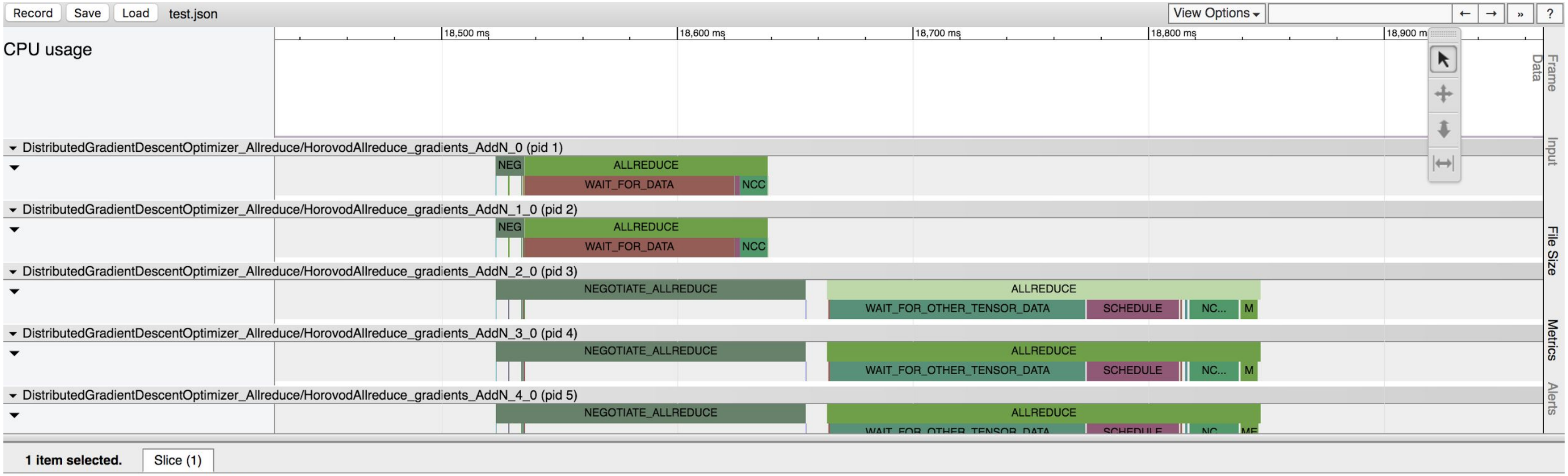
# Create the Estimator
mnist_classifier = tf.estimator.Estimator(
    model_fn=cnn_model_fn, model_dir="/tmp/mnist_convnet_model",
    config=tf.estimator.RunConfig(session_config=config))

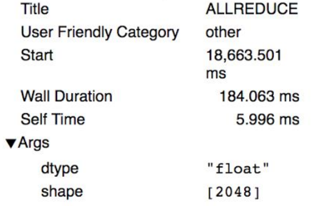
mnist_classifier.train(input_fn=train_input_fn, steps=100, hooks=hooks)
```

Running Horovod

- MPI takes care of launching processes on all machines
- Run on a 4 GPU machine (Open MPI 3.0.0):
  - ```
    $ mpirun -np 4 \
        -H localhost:4 \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH \
        python train.py
    ```
- Run on 4 machines with 4 GPUs (Open MPI 3.0.0):
  - ```
    $ mpirun -np 16 \
        -H server1:4,server2:4,server3:4,server4:4 \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH \
        python train.py
    ```
- Boilerplate `mpirun` arguments are easily hidden in a convenience script

Debugging - Horovod Timeline

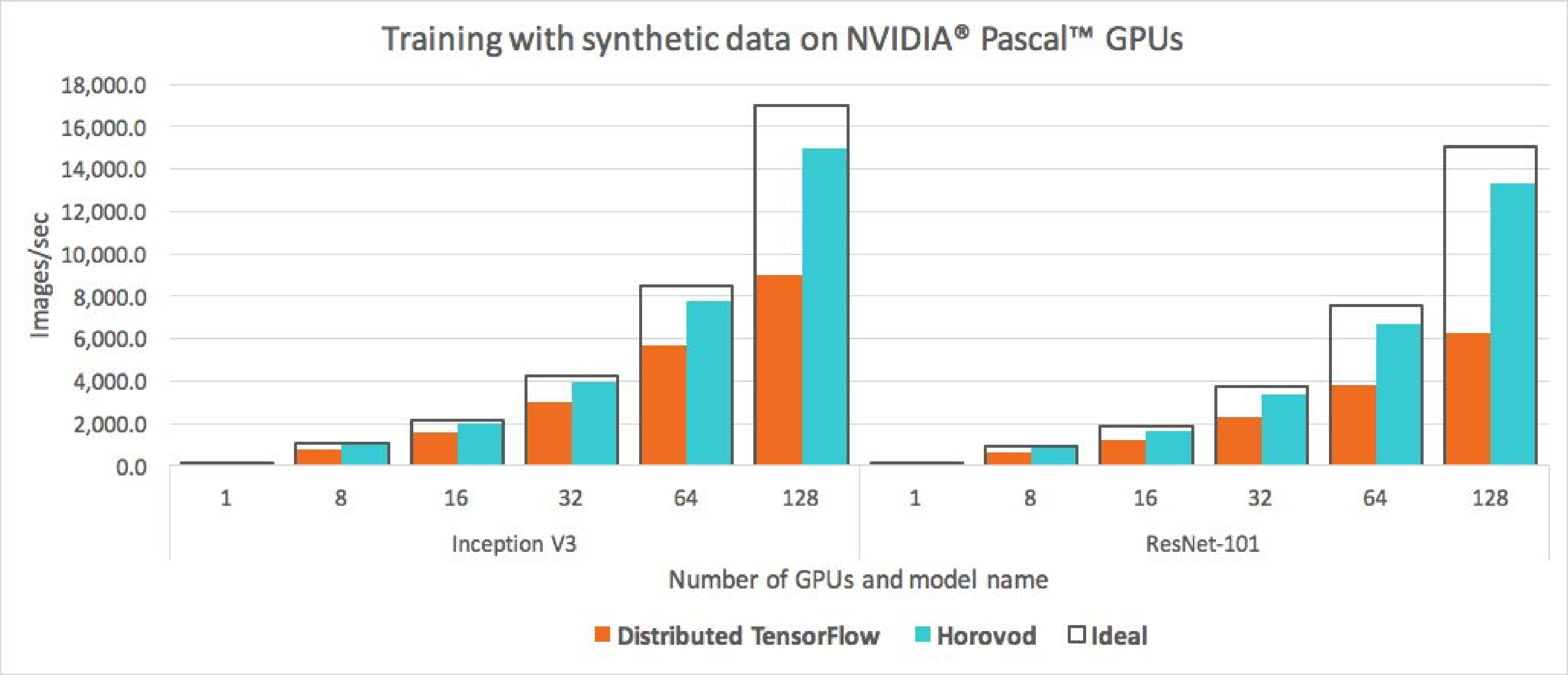




- Discovered that ResNet-152 has a lot of tiny tensors
- Added Tensor Fusion - smart batching causes large gains (bigger gain on less optimized networks)

UBER

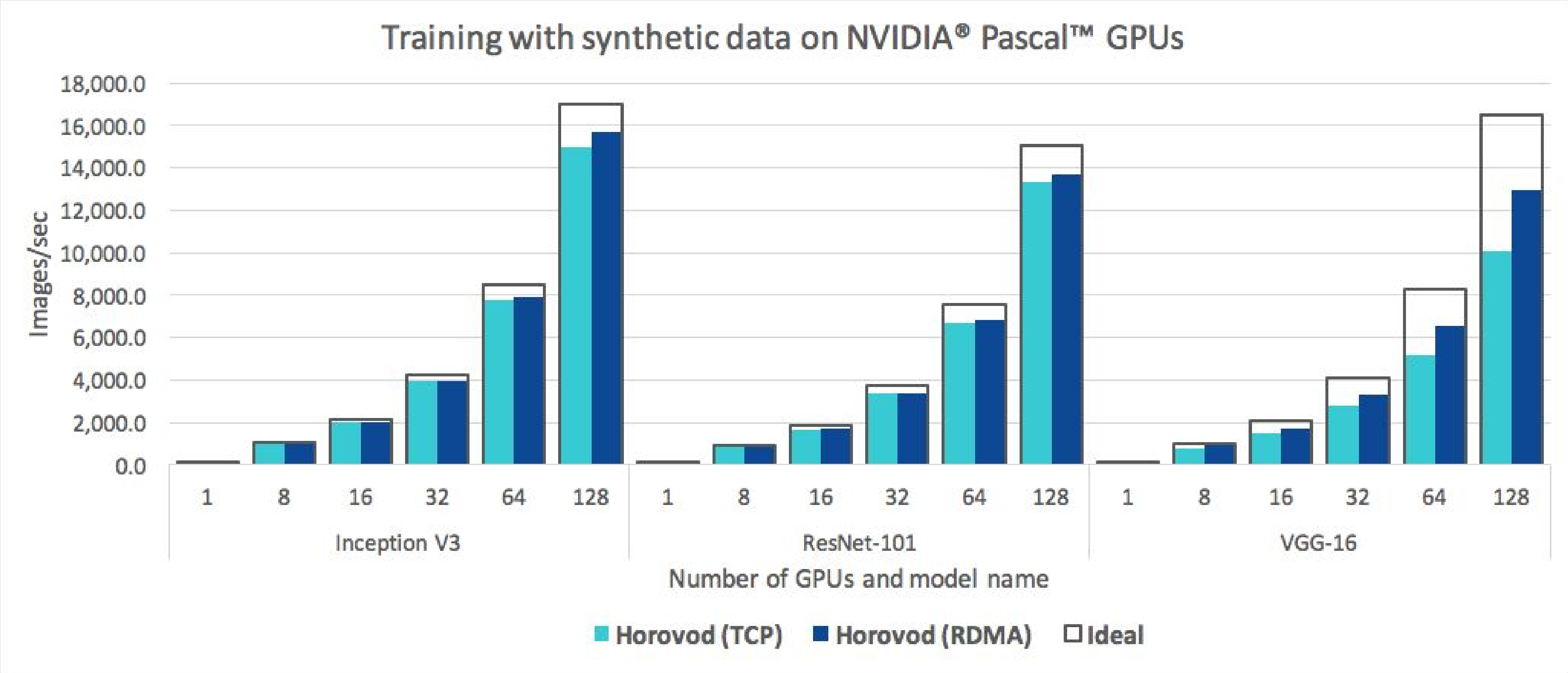
Horovod Performance



UBER

With Horovod, same ResNet-101 can be trained for one epoch on ImageNet in 1.5 minutes. Scaling efficiency is improved to 88%, making it twice as efficient as standard distributed TF.

Horovod Performance Cont.

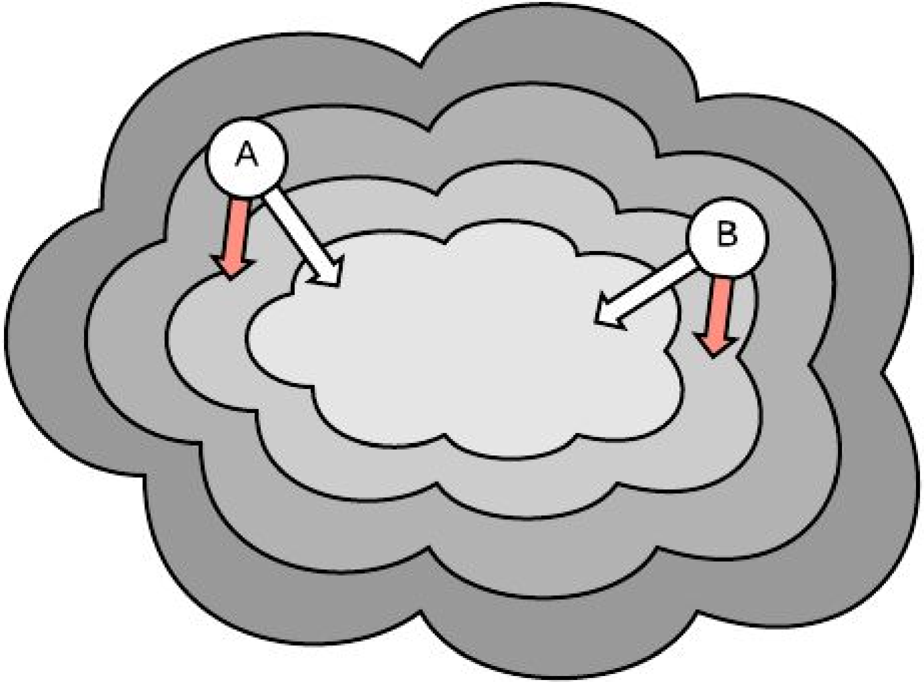


UBER

RDMA further helps to improve efficiency - by 30% for VGG-16.

Practical Aspects - Initialization

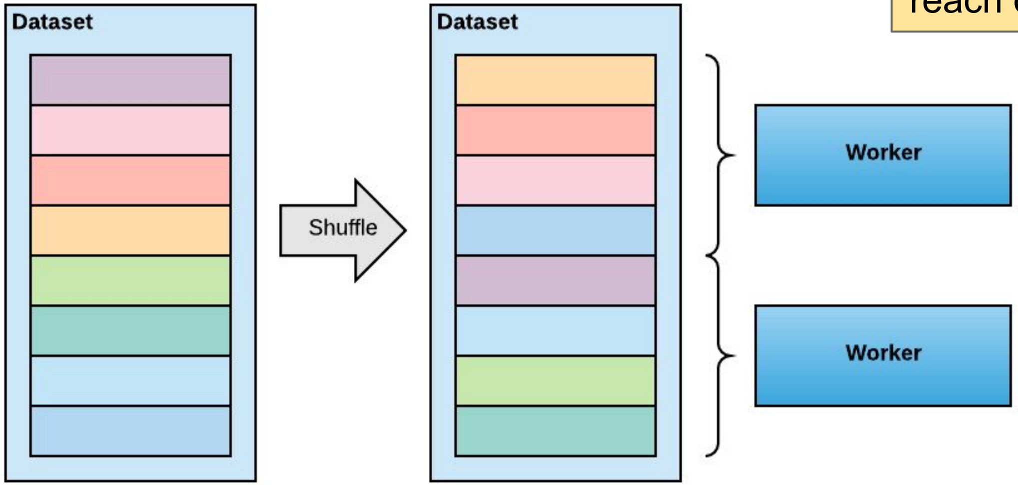
- Use broadcast operation to make sure all workers start with the same weights
- Otherwise, averaged gradient will not point towards minimum (shown in red)



Practical Aspects - Data Partitioning

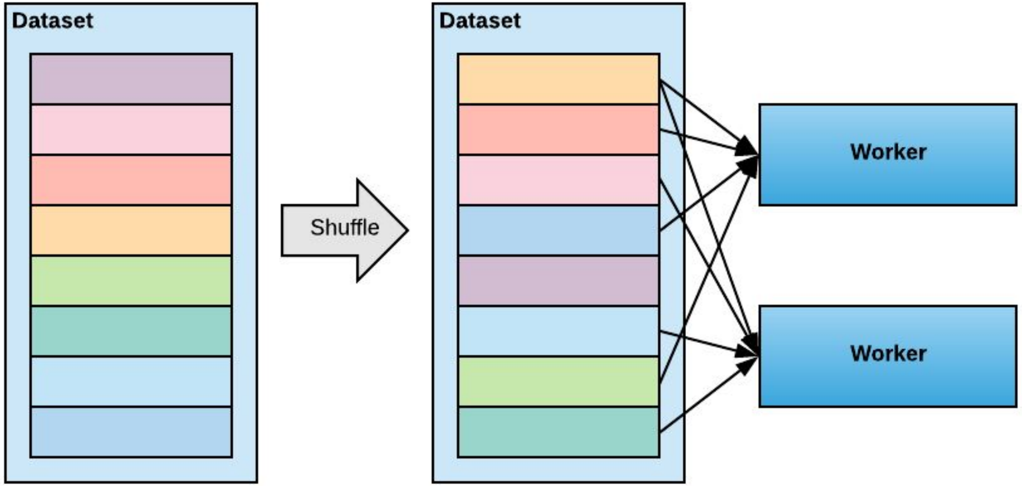
- Shuffle the dataset
- Partition records among workers
- Train by sequentially reading the partition
- After epoch is done, reshuffle and partition again

NOTE: make sure that all partitions contain the same number of batches, otherwise the training will reach deadlock



Practical Aspects - Random Sampling

- Shuffle the dataset
- Train by randomly reading data from whole dataset
- After epoch is done, reshuffle

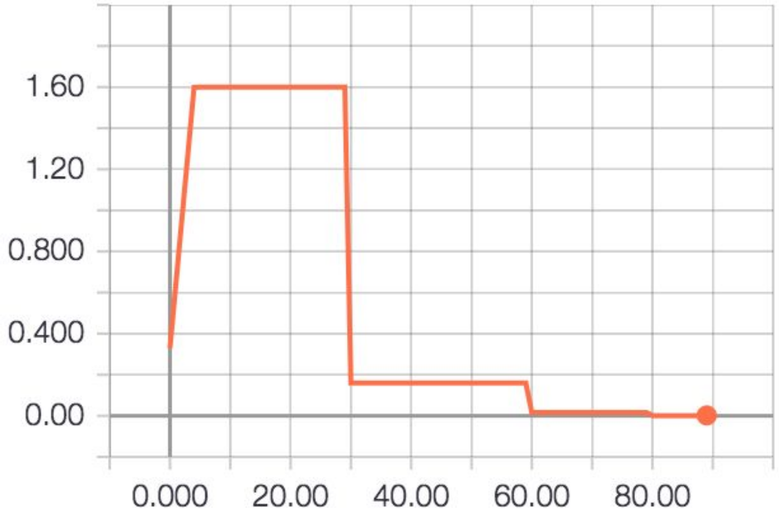


Practical Aspects - Data

- Random sampling may cause some records to be read multiple times in a single epoch, while others not read at all
- In practice, both approaches typically yield same results
- **Conclusion**: use the most convenient option for your case
- **Remember**: validation can also be distributed, but need to make sure to average validation results from all the workers when using learning rate schedules that depend on validation
  - Horovod comes with **`MetricAverageCallback`** for Keras

Practical Aspects - Learning Rate Adjustment

- Facebook in paper "[Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour]" ([arxiv.org/abs/1706.02677]) recommends linear scaling of learning rate:
    - $LR_N = LR_1 * N$
    - Requires smooth warmup during first K epochs, as shown below
    - Works up to batch size 8192
- Horovod comes with **`LearningRateWarmupCallback`** for Keras



Practical Aspects - Learning Rate Adjustment Cont.

- Yang You, Igor Gitman, Boris Ginsburg in paper “Large Batch Training of Convolutional Networks” demonstrated scaling to batch of 32K examples (arxiv.org/abs/1708.03888)
  - Use per-layer adaptive learning rate scaling
- Google published a paper “Don't Decay the Learning Rate, Increase the Batch Size” (arxiv.org/abs/1711.00489) arguing that typical learning rate decay can be replaced with an increase of the batch size

Practical Aspects - Checkpointing & Logs

- Typically, a server would have multiple GPUs
- To avoid clashes, write checkpoints, TensorBoard logs and other artifacts on worker 0:
  - ```
    if hvd.rank() == 0:
      # write checkpoint
    ```
    ```

Practical Results at Uber

- Used Facebook’s learning rate adjustment technique
- Trained convolutional networks and LSTMs in hours instead of days or weeks with the same final accuracy
- You can do that, too!

Giving Back

Horovod is available on GitHub:

https://github.com/uber/horovod

Thank you!

Horovod on our Eng Blog: https://eng.uber.com/horovod
Michelangelo on our Eng Blog: https://eng.uber.com/michelangelo
ML at Uber on YouTube: http://t.uber.com/ml-meetup

UBER