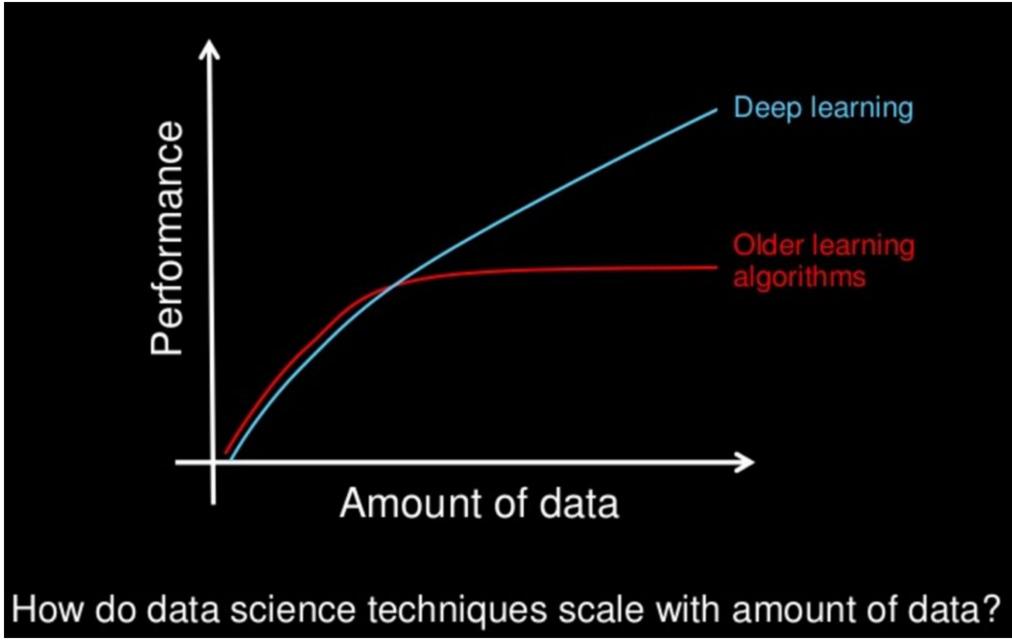
Distributed Deep Learning with Horovod

Alex Sergeev, Machine Learning Platform, Uber Engineering
@alsrgv

Uber

Deep Learning

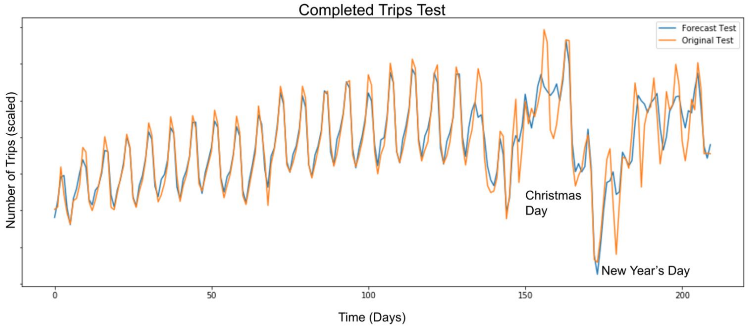
- Continues to improve accuracy long after older algorithms reach data saturation
- State of the art for vision, machine translation, and many other domains
- Capitalize on massive amount of research happening in the global community



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

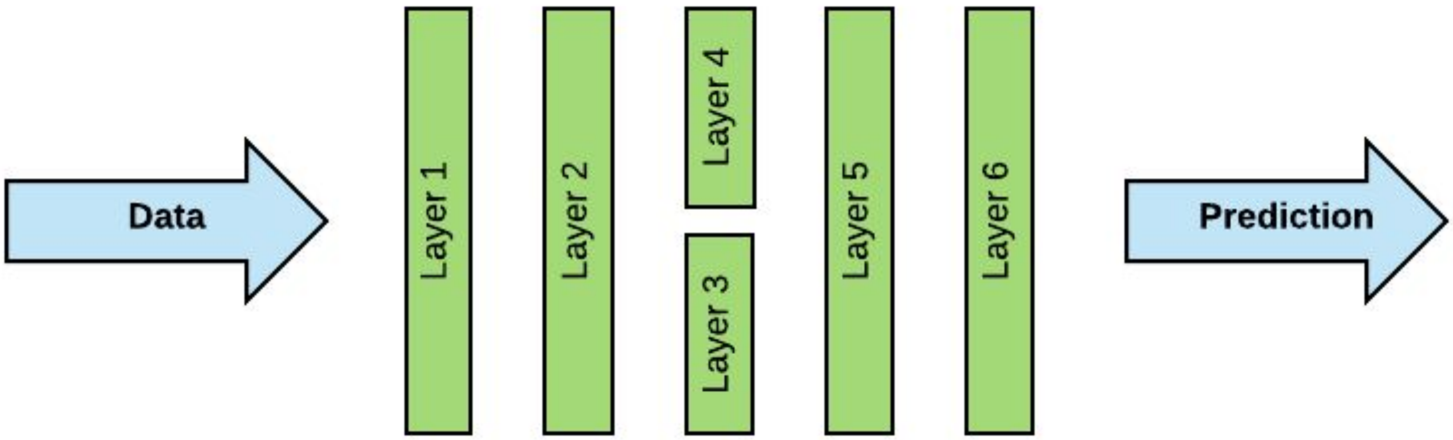
Deep Learning @ Uber

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



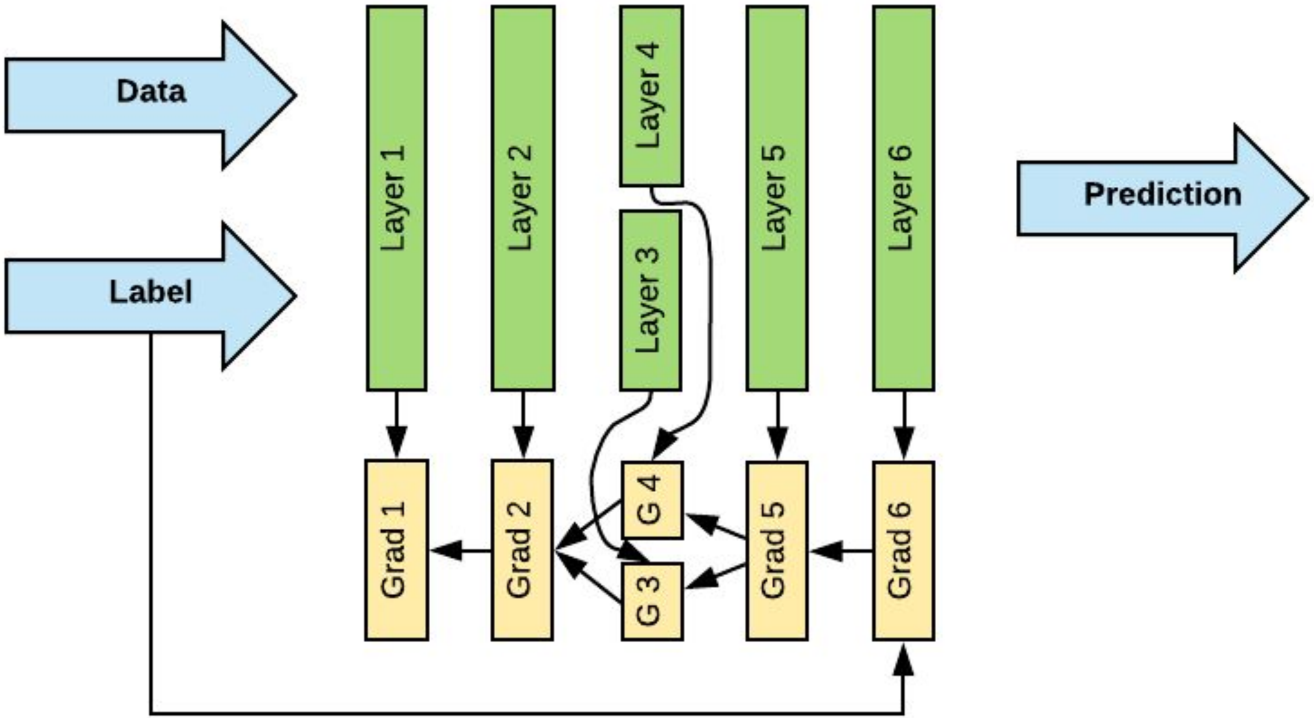
Uber

How does Deep Learning work?



Uber

How does Deep Learning training work?



Uber

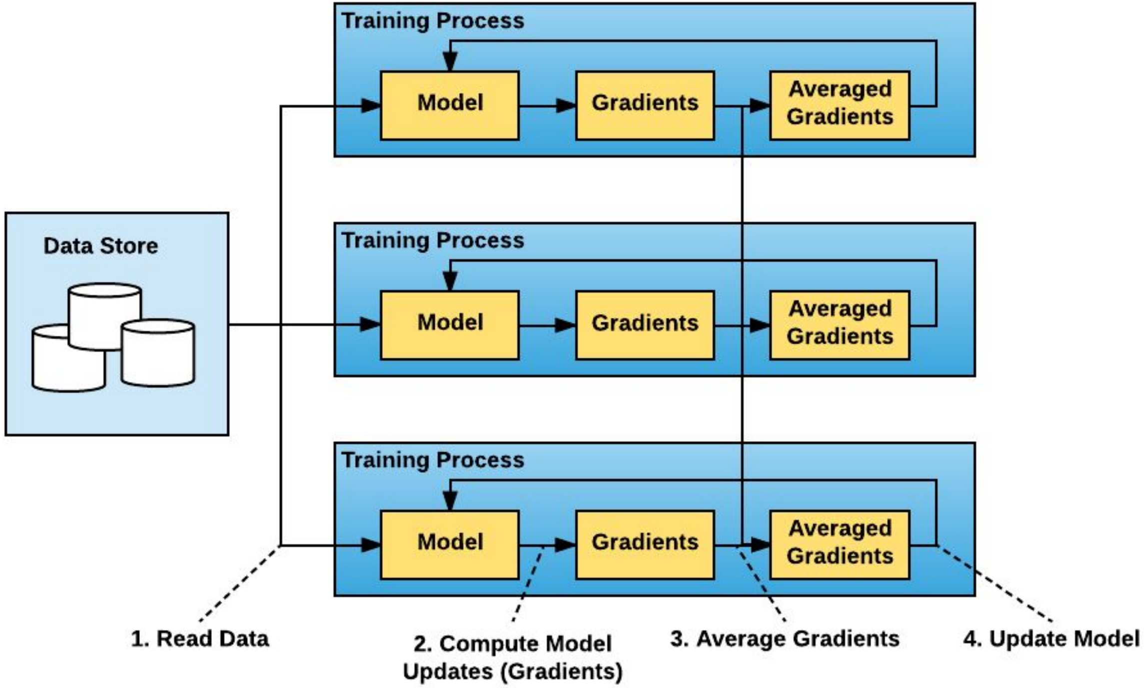
Massive amounts of data...

...make things slow (weeks!)

Solution:
distributed training.

How much GPU memory?
AWS p3x16large: 128GB
NVIDIA DGX-2: 512GB

Most models fit in a server.
→ Use data-parallel training.



Uber

Goals

There are many ways to do data-parallel training.
Some are more confusing than others. UX varies greatly.

Our goals:

1. Infrastructure people (like me ☺) deal with choosing servers, network gear, container environment, default containers, and tuning distributed training performance.

2. ML engineers focus on making great models that improve business using deep learning frameworks that they love.

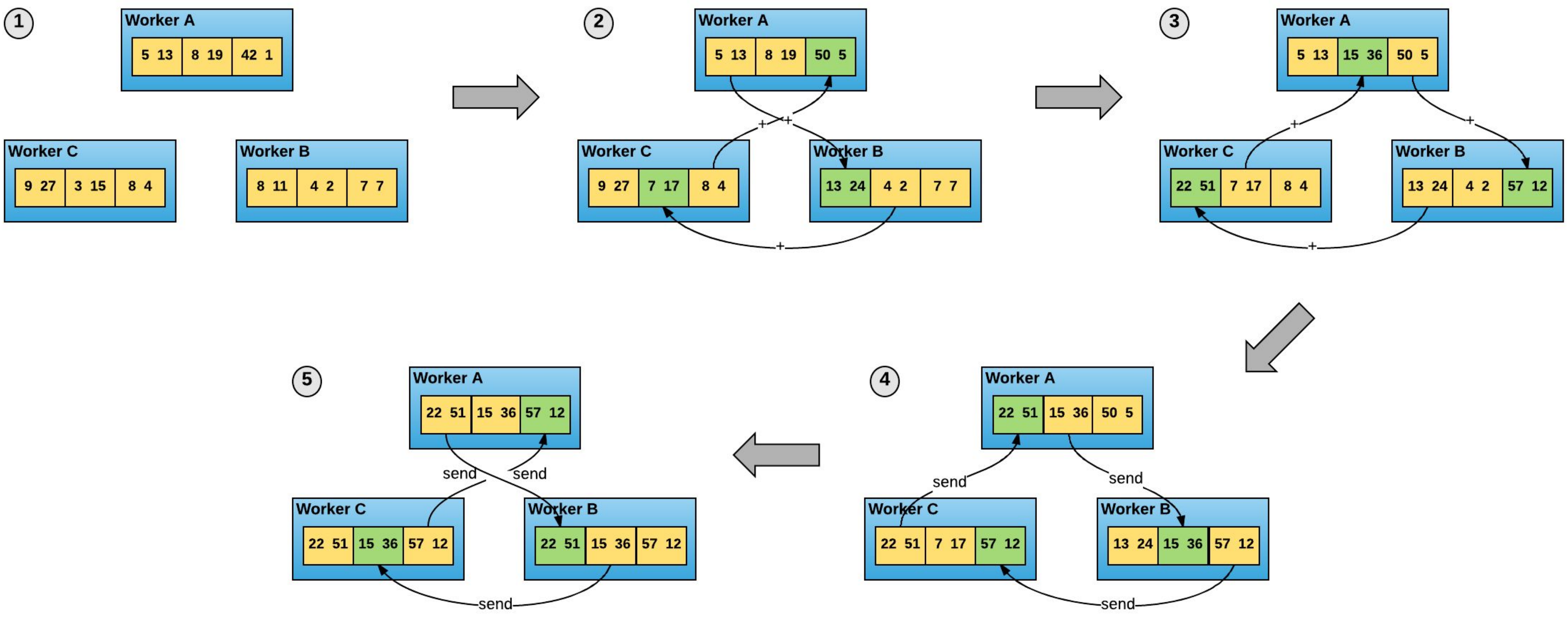
Meet Horovod

- Library for distributed deep learning.
- Works with stock TensorFlow, Keras, PyTorch, and Apache MXNet.
- Installs on top via `pip install horovod`.
- Uses advanced algorithms & can leverage features of high-performance networks (RDMA, GPUDirect).
- Separates infrastructure from ML engineers:
    - Infra team provides container & MPI environment
    - ML engineers use DL frameworks that they love
    - Both Infra team and ML engineers have consistent expectations for distributed training across frameworks

HOROVOD

horovod.ai

Horovod Technique: Ring-Allreduce

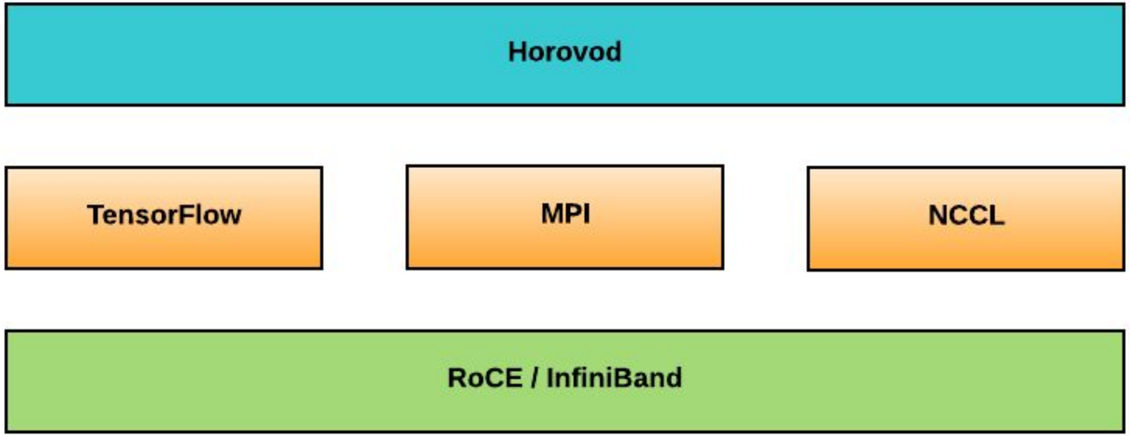


Patarasuk, P., & Yuan, X. (2009). Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2), 117-124. doi:10.1016/j.jpdc.2008.09.002

Uber

Horovod Stack

- Plugs into TensorFlow, Keras, PyTorch, and Apache MXNet via custom ops
- Uses MPI for worker discovery and reduction coordination
- Uses NVIDIA NCCL for actual reduction on the server and across servers



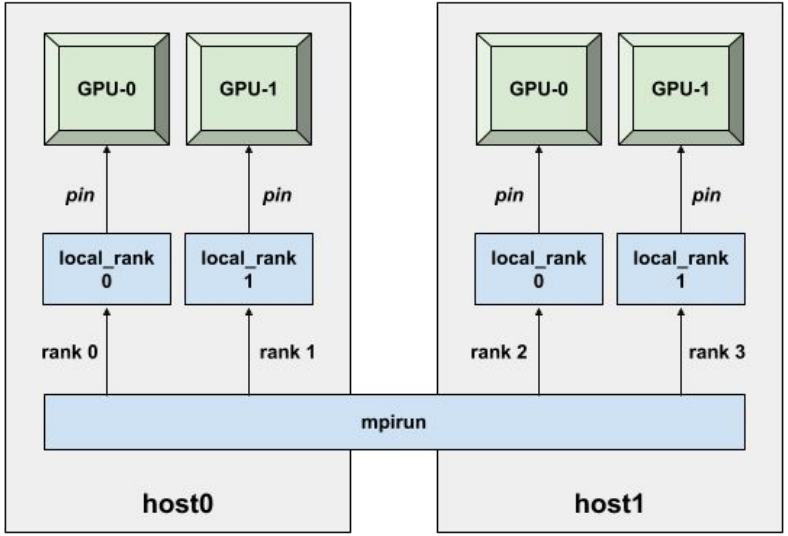
Using Horovod

#1. Initialize the library

```
import horovod.tensorflow as hvd

hvd.init()
```

#2. Pin GPU to be used

```
config = tf.ConfigProto()

config.gpu_options.visible_device_list = str(hvd.local_rank())
```

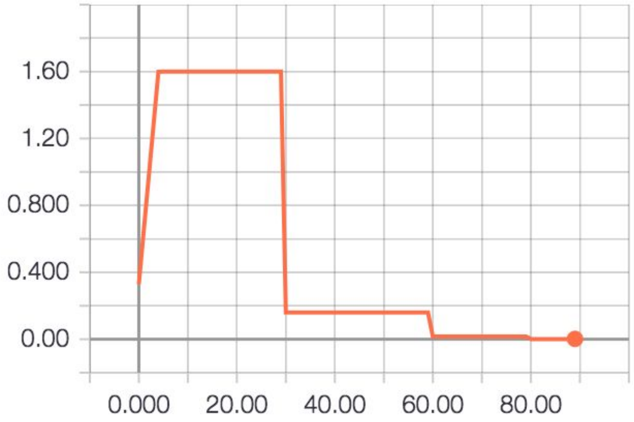


Uber

#3. Adjust LR & add Distributed Optimizer

```
opt = tf.train.MomentumOptimizer(lr=0.01 * hvd.size())
opt = hvd.DistributedOptimizer(opt)
```

- Facebook paper:
    - *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*
        - arxiv.org/abs/1706.02677
- Recommend linear scaling of learning rate:
    - $LR_N = LR_1 * N$
    - Smooth warm-up for the first K epochs
- Use `LearningRateWarmupCallback` for Keras



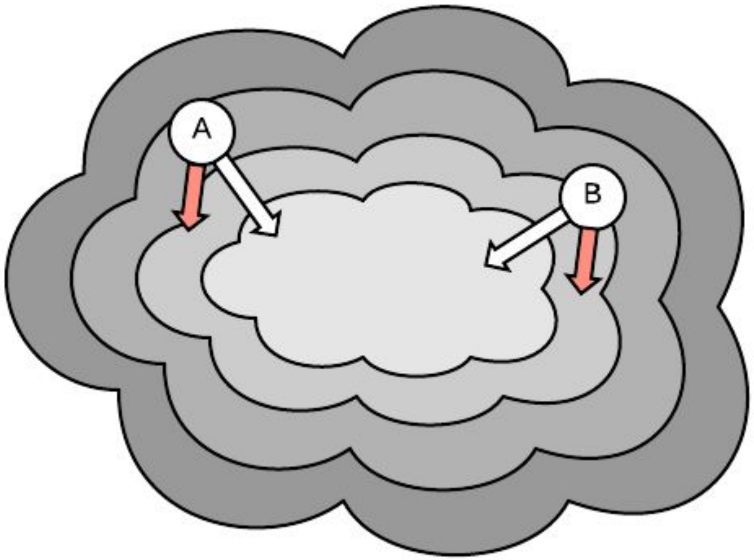
Uber

#3. 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

#4. Synchronize initial state

```
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
with tf.train.MonitoredTrainingSession(hooks=hooks, …) as mon_sess:
  …
# Or
bcast_op = hvd.broadcast_global_variables(0)
sess.run(bcast_op)
```



Uber

#5. Use checkpoints only on first worker

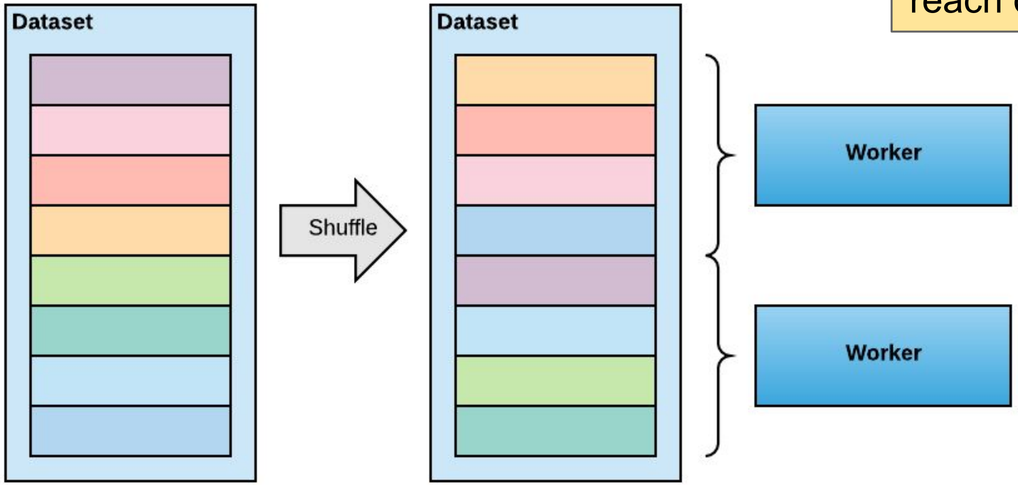
```
ckpt_dir = "/tmp/train_logs" if hvd.rank() == 0 else None

with tf.train.MonitoredTrainingSession(checkpoint_dir=ckpt_dir, …) as mon_sess:

  …
```

Uber

#6. Data: Partitioning

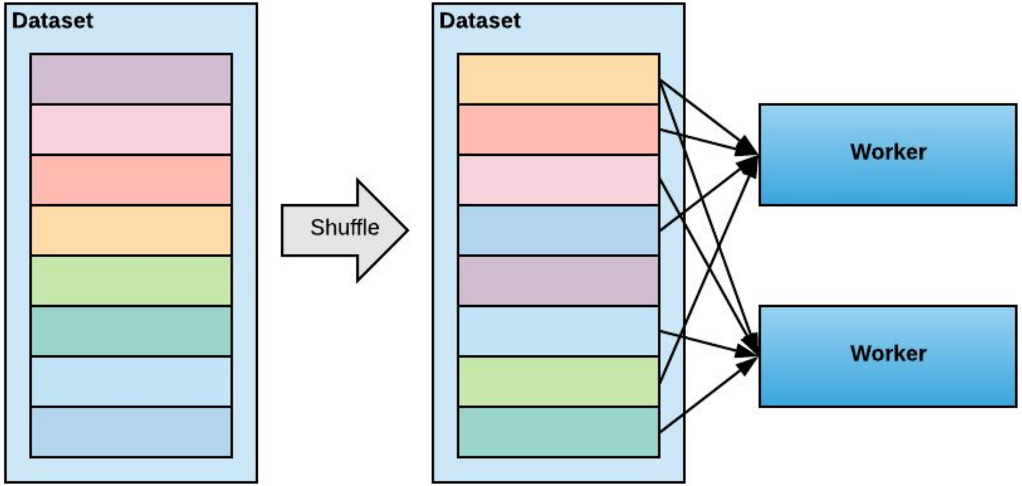
- Shuffle the dataset
- Partition records among workers
- Train by sequentially reading the partition
- After epoch is done, reshuffle and partition again

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



#6. Data: Random Sampling

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



Uber

#6. Data Review

- 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

Full 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.MomentumOptimizer(
  lr=0.01 * hvd.size())

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

# Add hook to synchronize initial state
hooks =[hvd.BroadcastGlobalVariablesHook(0)]

# Only checkpoint on rank 0
ckpt_dir = "/tmp/train_logs" \
  if hvd.rank() == 0 else None

# 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=ckpt_
dir, config=config, hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Uber

There's more…

Horovod for TensorFlow

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

Horovod for All

```
import horovod.tensorflow as hvd
import horovod.keras as hvd
import horovod.tensorflow.keras as hvd
import horovod.torch as hvd
import horovod.mxnet as hvd
# more frameworks coming
```

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
config = tf.ConfigProto()
config.gpu_options.visible_device_list =
  str(hvd.local_rank())
K.set_session(tf.Session(config=config))

# Build model…
model = ...
opt = keras.optimizers.Adadelta(
  lr=1.0 * hvd.size())

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

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

# Broadcast initial variable state.
callbacks =
[hvd.callbacks.BroadcastGlobalVariablesCallback(0)]

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

Uber

TensorFlow Eager Mode

```
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())

tf.enable_eager_execution(config=config)

# Adjust learning rate based on number of GPUs.
opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
```

```
for batch, (images, labels) in enumerate(dataset):
  with tf.GradientTape() as tape:
    loss = …

    # Broadcast model variables
    if batch == 0:
      hvd.broadcast_variables(0, model.variables)

    # Add DistributedGradientTape
    tape = hvd.DistributedGradientTape(tape)

    grads = tape.gradient(loss_value, model.variables)
    opt.apply_gradients(zip(grads, model.variables))
```

Uber

PyTorch

```
import torch
import horovod.torch as hvd

# Initialize Horovod
hvd.init()

# Horovod: pin GPU to local rank.
torch.cuda.set_device(hvd.local_rank())

# Build model.
model = Net()
model.cuda()
optimizer = optim.SGD(model.parameters())

# Wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
  optimizer,
  named_parameters=model.named_parameters())

# Horovod: broadcast parameters.
hvd.broadcast_parameters(
  model.state_dict(),
  root_rank=0)

for epoch in range(100):
  for batch_idx, (data, target) in …:
    optimizer.zero_grad()
    output = model(data)
    loss = F.nll_loss(output, target)
    loss.backward()
    optimizer.step()
```

Uber

Apache MXNet

```
import torch
import horovod.mxnet as hvd

# Initialize Horovod
hvd.init()

# Horovod: pin GPU to local rank.
context = mx.gpu(hvd.local_rank())

# Build model.
net = …
loss = ...
model = mx.mod.Module(symbol=loss, context=context)

# Wrap optimizer with DistributedOptimizer.
opt = hvd.DistributedOptimizer(opt)

# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.get_params(),
root_rank=0)

model.fit(...)
```

Uber

Running Horovod

Single-node:

```
$ horovodrun -np 4 -H localhost:4 python train.py
```

Multi-node:

```
$ horovodrun -np 16 -H server1:4,server2:4,server3:4,server4:4 python train.py
```

Running Horovod: Under the Hood

- MPI takes care of launching processes on all machines
- Run on a 4 GPU machine:

```
$ mpirun -np 4 \
    -H localhost:4 \
    -bind-to none -map-by slot \
    -mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include eth0 \
    -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x ... \
    python train.py
```

- Run on 4 machines with 4 GPUs:

```
$ mpirun -np 16 \
    -H server1:4,server2:4,server3:4,server4:4 \
    -bind-to none -map-by slot \
    -mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include eth0 \
    -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x ... \
    python train.py
```

Horovod on Spark

```
In [1]: from pyspark import SparkConf
        from pyspark import SparkContext
        import horovod.spark

In [2]: sc = SparkContext(conf=SparkConf())

In [3]: def train():
            import horovod.tensorflow as hvd
            hvd.init()
            return hvd.rank()

In [4]: print(horovod.spark.run(train, num_proc=4))

        [0, 1, 2, 3]

In [ ]:
```

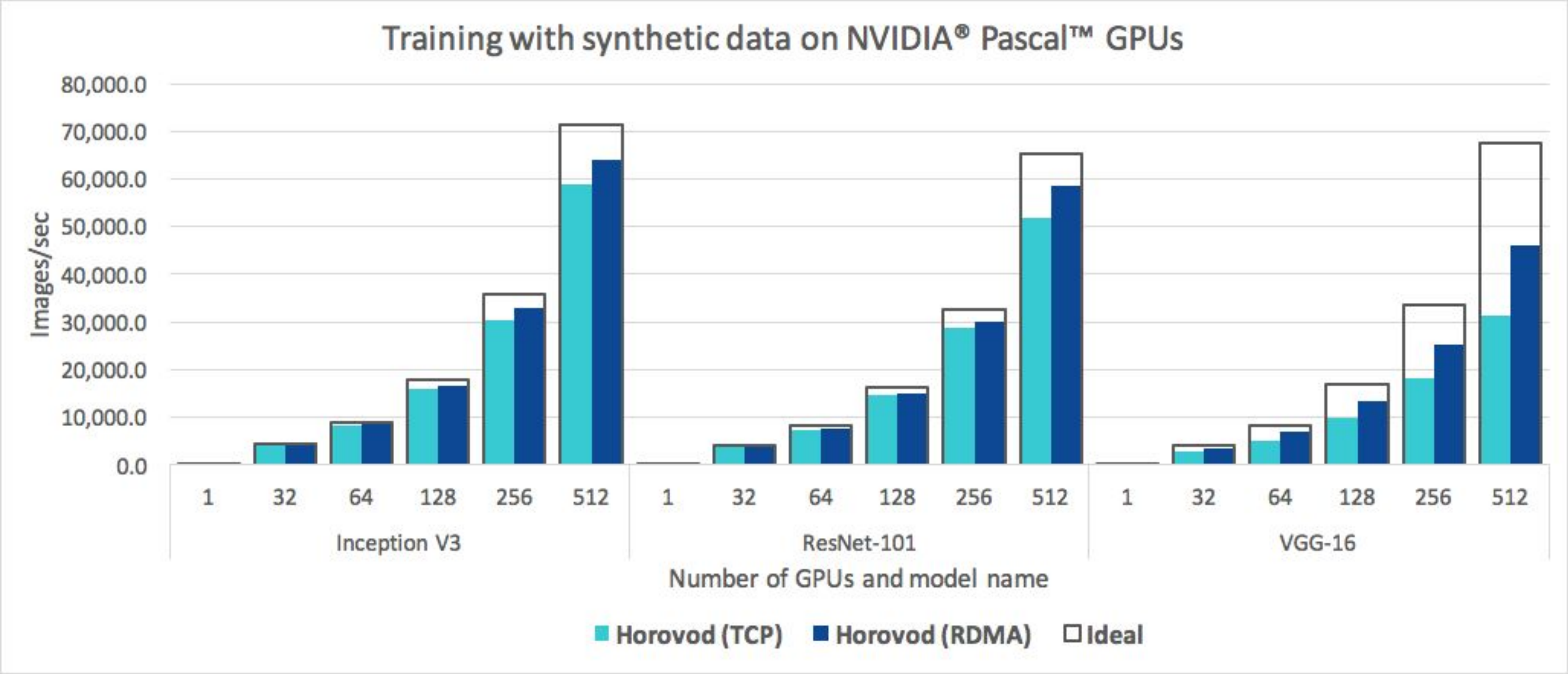
Uber

Why Spark?

- Allows users to leverage existing Spark infrastructure
  - Including Jupyter and IPython!
- Data preparation & model training in the same environment
- Save to Parquet and use Petastorm for data ingestion
  - Takes care of random shuffling, fault tolerance, etc
  - https://github.com/uber/petastorm

Uber

Horovod Performance



Horovod scales well beyond 128 GPUs. RDMA helps at a large scale, especially to small models with fully-connected layers like VGG-16, which are very hard to scale.

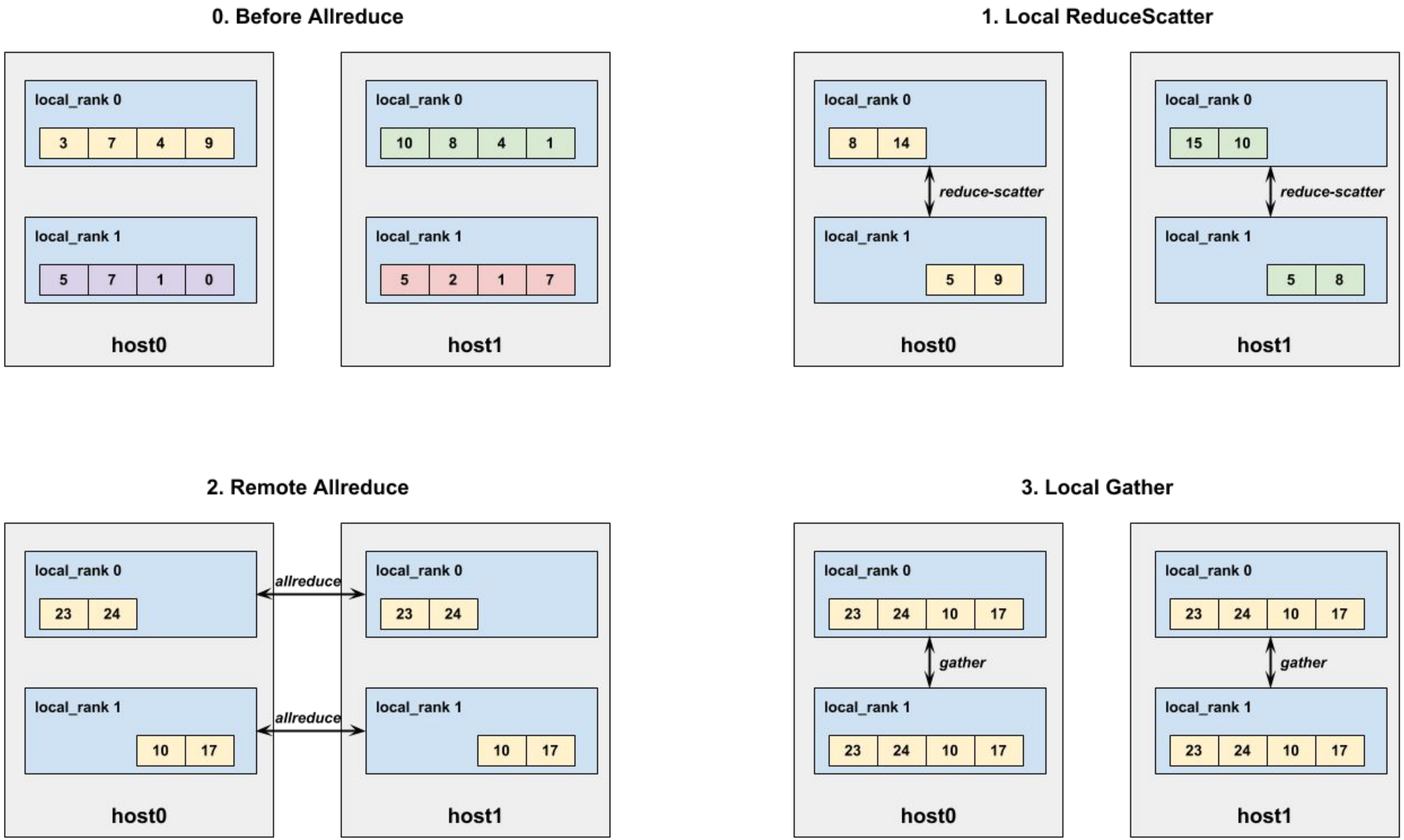
Horovod Knobs: Hierarchical Algorithms

```
$ HOROVOD_HIERARCHICAL_ALLREDUCE=1 horovodrun ...
$ HOROVOD_HIERARCHICAL_ALLGATHER=1 horovodrun ...
```

- Contributed by NVIDIA & Amazon
- First allreduce locally, then allreduce across nodes in parallel
  - Each worker responsible for a different chunk of the buffer
- Speeds up training for very large cluster setups
  - Homogenous nodes (same # GPUs)
  - Many GPUs per node

Uber

Hierarchical Allreduce: Example



Uber

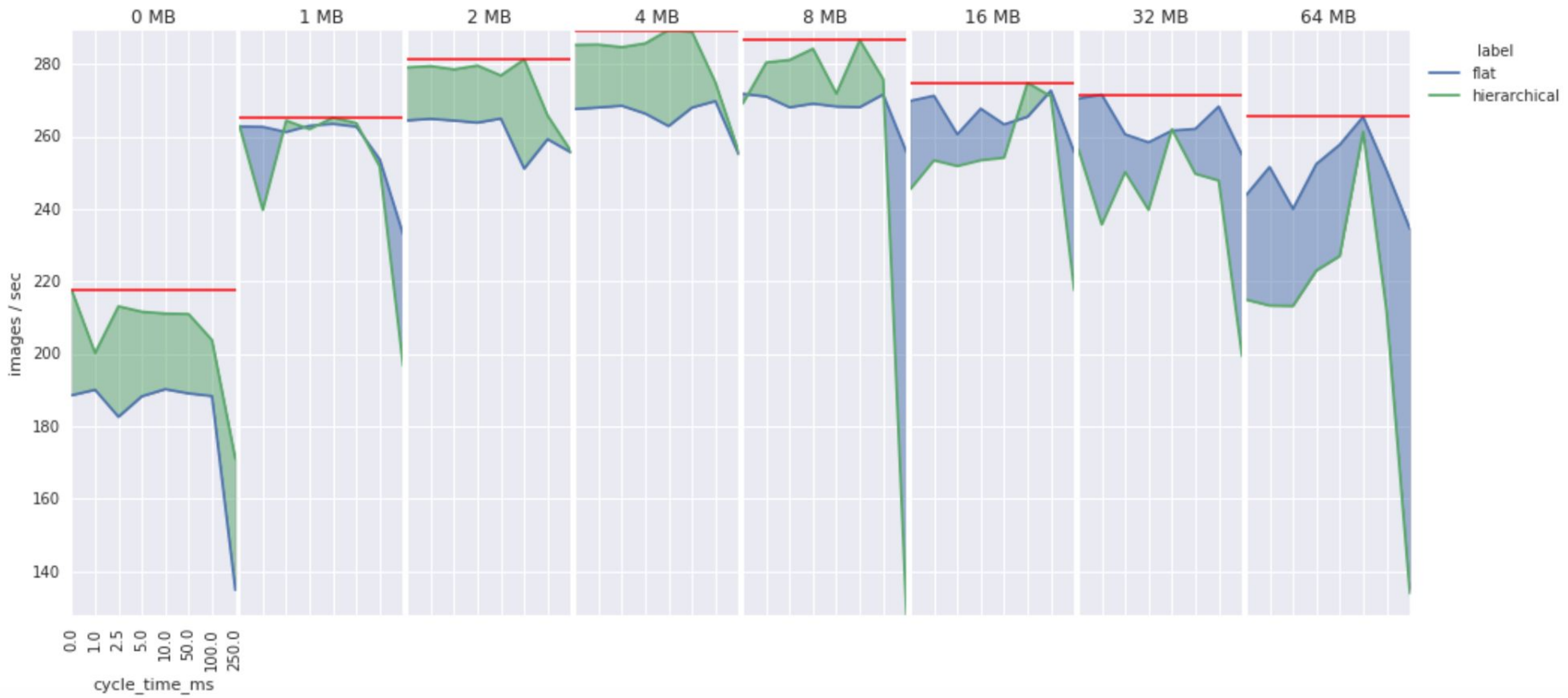
Horovod Knobs: Tensor Fusion

```
$ HOROVOD_FUSION_THRESHOLD=67108864 HOROVOD_CYCLE_TIME=5 horovodrun ...
```

- Batch tensors together during allreduce
- **Fusion Threshold:** size of batching buffer (in bytes)
- **Cycle Time:** wait time between sending batches (in milliseconds)

Uber

Horovod Knobs: Auto Tuning with Bayesian Optimization



Uber

Use **HOROVOD_AUTOTUNE=1** to find the best Horovod parameters

Horovod Knobs: Gradient Compression

- FP16 allreduce
  - `hvd.DistributedOptimizer(..., compression=hvd.Compression.fp16)`
  - Reduces arithmetic computation on GPU
  - Reduces network utilization
- Not auto-selected by Auto-tuning since it may affect model convergence
- More techniques coming - contribution welcome!

Practical Results at Uber and beyond

- Horovod is accepted as the only way Uber does distributed deep learning
- We train both convolutional networks and LSTMs in hours instead of days or weeks with the same final accuracy - game changer
- Horovod is widely used by various companies including NVIDIA, Amazon and Alibaba and various research institutions
- Horovod is included in various deep learning distributions:
  AWS Deep Learning AMI, GCP Deep Learning VM, Azure Data Science VM, NVIDIA GPU Cloud, IBM FfDL, Databricks Runtime, IBM Watson Studio

Thank you!

http://horovod.ai

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